

Ryu-kontrollerin ja Snort-tunkeilijan estojärjestelmän integraatio

Maria Ylitalo

Opinnäytetyö

Toukokuu 2016

Tekniikan ja liikenteen ala

Insinööri (AMK) ICT, Tietotekniikan koulutusohjelma

Tietoverkkotekniikka

Tekijä(t) Ylitalo, Maria	Julkaisun laji Opinnäytetyö	Päivämäärä 05 2016
	Sivumäärä 104	Julkaisun kieli Suomi
		Verkojulkaisulupa myönnetty: (X)
Työn nimi Ryu-kontrollerin ja Snort-tunkeilijan estojärjestelmän integraatio		
Tutkinto-ohjelma Tietotekniikan koulutusohjelma		
Työn ohjaaja(t) Jarmo Viinikanoja Mika Rantonen		
Toimeksiantaja(t) Cyber Trust JAMK, Janne Alatalo		
Tiivistelmä <p>Digilen haluaa yhteistyökumppaniensa kanssa parantaa suomalaista tietoturvateollisuutta maailmanmarkkinoilla. Sen vuoksi he ovat luoneet CyberTrust-ohjelman, joka tutkii uusia verkkoratkaisuja ja kehittää tietoturvahankkeita. CyberTrust on keskittynyt tutkimaan muun muassa SDN-verkkotekniikan mahdollisuuksia.</p> <p>Opinnäytetyön tavoitteena oli tutkia Ryu-kontrollerin ja Snort-tunkeilijan estojärjestelmän välistä integraatiota. Ensimmäinen testi integraation parissa osoitti, että integraatio oli puutteellinen ja sen kehittäminen oli mitä ilmeisemmin jäänyt kesken. Työn aikana integraatiota kehitettiin niin, että sen avulla pystyttiin ajamaan yksinkertaisia testejä. Näiden testien avulla pystyttiin varmentamaan integraation toiminta ja tulosten perusteella arvioitiin, onko integraatiolla paikkaa SDN-verkossa.</p> <p>Integraatio sisälsi monia ongelmia. Yksi merkittävimmistä ongelmista oli integraation hitaus. Aggressiivisessä hyökkäyksessä integraatio pärjäsi, jos hyökkäys kohdistui vain yhdeltä laitteelta. Tutkimustulosten analysoinnin ja sitä edeltävien johtopäätelmien vuoksi voitiin sanoa, ettei integraatio nykyisellään ole toimiva osa SDN-verkon tietoturvaa. Integraatiolla oli kuitenkin paljon potentiaalia, joten sen jatkokehittäminen olisi kannattavaa. Kehittämällä ja parantamalla integraation toimintaa, siitä voisi saada hyvän ja tehokkaan työkalun edistämään SDN-verkon tietoturvaa.</p>		
Avainsanat (asiasanat) SDN, Ryu, Snort, Cyber Trust		
Muut tiedot		

Author(s) Ylitalo, Maria	Type of publication Bachelor's Thesis	Date 05 2016 Language of publication: Finnish
	Number of pages 104	Permission for web publication: (X)
	Title of publication Integration of Ryu controller and Snort intrusion prevention system	
Degree programme Data Network Technology		
Supervisor(s) Jarmo Viinikanoja Mika Rantonen		
Assigned by Cyber Trust JAMK, Janne Alatalo		
Abstract <p>Digile with its collaborators wants to improve Finnish cyber technology on global markets. For this purpose, they have created CyberTrust program that researches new network solutions and develops network security. CyberTrust has focused their researches on possibilities of SDN network, for example.</p> <p>The goal of the thesis was to explore the integration of Ryu controller and Snort intrusion prevention system. The first experiment with integration showed that the integration was incomplete and its development was interrupted. During the thesis project the integration was developed further so that new simple experiments could be made. With the help of these new experiments the functionality of integration was guaranteed. Using the results of experiments it could be evaluated if there was any place for integration in SDN network.</p> <p>The integration presented many issues one of which with significance was the slowness of integration. In the case of an aggressive attacker the integration worked well if the attacker used only one device. After analyzing the new results a conclusion could be drawn that the integration as implemented here does not improve the security of SDN networks; however integration had great potential and its development in future was recommended. The development and improvement of the integration could make it an effective tool to improve the security of SDN network.</p>		
Keywords/tags (subjects) SDN, Ryu, Snort, Cyber Trust		
Miscellaneous		

Sisältö

	Termit ja lyhenteet	9
1	Johdanto.....	13
	1.1 Työn tavoitteet	13
2	SDN-verkkotekniikka	14
	2.1 SDN-verkon arkkitehtuuri.....	14
	2.2 Virtualisointi tehostaa resurssien käyttöä	16
3	Ryu-kontrolleri.....	17
	3.1 Ryun arkkitehtuuri.....	17
	3.2 OpenFlow	19
	3.3 Ryun toiminta perustuu tapahtumiin.....	21
	3.4 Kontrollerien vertailua	22
4	Tietoturva	24
	4.1 Tietoturva SDN-tekniikassa	24
	4.2 Snort kasvattaa tietoverkon tietoturvaa	25
	4.3 DoS ja Man in the Middle -hyökkäys.....	26
5	OSI-malli	27
6	Snort.....	29
	6.1 Snortin ominaisuudet	30
	6.2 Snortin komponentit	31
	6.2.1 Preprocessors – Esikäsittelijät	32
	6.2.2 Detection Engine	34
	6.2.3 Output-moduuli.....	35
	6.3 Snort-säännön rakenne	39
	6.3.1 Sisällön vaihtoehdot	41
7	Työn toteutus	42
	7.1 Asennus	43
	7.1.1 Ryu	43

7.1.2	Mininet	45
7.1.3	Snort	46
7.2	Ensimmäinen testi	51
7.3	Integraation tarkastelu	57
7.3.1	Applikaation lähdekoodin tarkastelu	57
7.3.2	Integraation kehittäminen.....	58
7.3.3	Tarvittavien lisäohjelmien asentaminen	60
7.3.4	Applikaation lähdekoodin kehittäminen	70
7.4	Toinen testi.....	76
7.5	Kolmas testi	79
7.5.1	Tulosten analysointi.....	81
7.5.2	Tulosten yhteenveto.....	84
8	Pohdinta	85
8.1	Integraation ongelmat.....	85
8.2	Tulosten luotettavuus	87
8.3	Applikaation jatkokehitys	88
	Lähteet.....	90
	Liitteet	93
	Liite 1. Simple_switch_snort.py	93
	Liite 2. Snort.py	97
	Liite 3. Snorby-tietokannan tietotauluja	103
	Liite 4. Kolmannen testin tulokset	104

Kuviot

Kuvio 1. Kontrollitason ja välitystason eriytys	15
Kuvio 2. SDN-verkkotekniikan kolme elementtiä.....	16
Kuvio 3. Ryun arkkitehtuuri.....	18
Kuvio 4. Vuotaulun toimintaperiaate.....	20
Kuvio 5. Ryun OpenFlow-viestit	21
Kuvio 6. Ryun ohjelmien arkkitehtuuri.....	22
Kuvio 7. SYN-flood hyökkäys	26
Kuvio 8. Man-in-the-middle hyökkäys, mies yhteyden välissä	27
Kuvio 9. OSI-malli	28
Kuvio 10. Snortin komponentit	32
Kuvio 11. Esimerkki liikenteen normaalisoinnista.....	33
Kuvio 12. Esimerkki Snort-hälytyksestä	36
Kuvio 13. Esimerkki Snortin loki-merkinnästä.....	37
Kuvio 14. Toinen esimerkki Snortin loki-merkinnästä	38
Kuvio 15. Snorby Dashboard	39
Kuvio 16. Snort säännön rakenne	40
Kuvio 17. Esimerkki Snort säännöstä	41
Kuvio 18. Toinen esimerkki Snort säännöstä	42
Kuvio 19. Ryu-applikaation onnistunut käynnistäminen	45
Kuvio 20. Mininet-verkon testaaminen	46
Kuvio 21. Snortin asennuksen varmistaminen.....	47
Kuvio 22. Snortin muuttujien lisääminen.....	49
Kuvio 23. Sääntöjen tiedostohakemiston lisääminen.....	50
Kuvio 24. Include -tiedostosijaintien lisääminen	50
Kuvio 25. Snort-konfiguraatitiedoston varmistaminen	51
Kuvio 26. Ensimmäisen testin verkkotopologia	52
Kuvio 27. Mininet testiverkon luominen.....	53
Kuvio 28. Unixsock-toiminnon asettaminen	54
Kuvio 29. Simple_switch_snort.py -applikaation suorittaminen	55
Kuvio 30. Pigrelay-applikaation muokkaaminen.....	56
Kuvio 31. Pigrelayn ja kontrollerin yhteyden varmentaminen	56

Kuvio 32. Hälytys Ryu-kontrollerilla	57
Kuvio 33. Root-käyttäjän lisääminen database.yml-tiedostoon.....	61
Kuvio 34. Snorby-käyttäjän lisääminen database.yml-tiedostoon	63
Kuvio 35. Snorbyn sisäänkirjautumisikkuna.....	64
Kuvio 36. Phusion Passenger asennusohjelman kopioitavat rivit.....	65
Kuvio 37. Passenger.conf-tiedosto.....	66
Kuvio 38. Passenger.conf-tiedosto.....	66
Kuvio 39 Snorby.conf-tiedosto.....	67
Kuvio 40. snort.py – dump-alert	71
Kuvio 41. Snort.py – get_snort_rule	73
Kuvio 42. Snort.py – packet_print.....	75
Kuvio 43. Snort.py – Send_flow_mod-luokka	76
Kuvio 44. Toisen testin säännöt.	76
Kuvio 45. Hälytys viesti Ryu-konsolilla	78
Kuvio 46. HTTP-hälytys Ryu-konsolilla	78
Kuvio 47. Ryu on luonut uudet vuosäännöt kytkimen S1 vuotauluun.....	79
Kuvio 48. Snort.py konfiguraation.....	80
Kuvio 49. Hping3	81
Kuvio 50. Hping3-hylkkäys nopeudella 100 pkt/sek	83
Kuvio 51. Hping3-hyökkäys nopeudella 10 000 pkt/sek	84
Kuvio 52. Iphdr-tietotaulu	103
Kuvio 53. Events_with_join-tietotaulu.....	103

Taulukot

Taulukko 1. Kontrollerin käyttötarkoitusten vertailu	24
Taulukko 2. Ryun vaatimat lisäosat.....	43
Taulukko 3. Mininet-komennon argumentit.....	53
Taulukko 4. Snort-komennon argumentit.....	55
Taulukko 5. Barnyard2-komennon argumentit.....	70
Taulukko 6. Tuple-objektin kentät	74
Taulukko 7. Hping3-hyökkäys nopeudella 100 pkt/sek	82
Taulukko 8. Hping3-hyökkäys nopeudella 10 000 pkt/sek	83
Taulukko 9. OpenFlow 1.3 Match-kentät	86
Taulukko 10. Yhteenveto kolmannen testin tuloksista.....	104

Termit ja Lyhenteet

API	Ohjelmointirajapinta (Application Programming Interface). Määritelmä, jonka mukaan eri ohjelmat voivat keskustella keskenään lähettämällä toisilleen pyyntöjä tai tapahtumia.
ARP	Address Resolution Protocol. ARP-protokollan avulla selvitetään verkkolaitteiden IP-osoitetta vastaava MAC-osoite
Avoin lähdekoodi	eng. Open Source. Yleisesti saatavilla ja kaikkien käyttäjien muokattavissa oleva ohjelman lähdekoodi. Vertaa kaupallinen ohjelma.
Barnyard2	Ohjelmisto, joka kykenee muuttamaan binäärimuotoista dataa luettavaan muotoon ja tallettamaan sen useaan eri kohteeseen.
Beacon	Java pohjainen SDN-verkon kontrolleri
CAM-taulu	Content Addressable Memory. MAC-välitystaulu
DAQ	Data Acquisition-kirjasto. Snortin medialta datan tiedonkeräämiseen vaadittava lisäosa
Detection Engine	Snort-ohjelman komponentti, joka tekee paljon prosessoriresurssia vaativaa pakettianalyysiä.
DNS	Domain name server. Nimipalvelin, joka muuttaa selkeäkielisen nimen IP-osoitteeksi ja toisinpäin.
DoS	Denial of Service eli palvelunestohyökkäys.
DPID	Datapath ID. Yksilöivä arvo, jonka avulla SDN-kontrolleri tunnistaa SDN-verkon kytkimet
False negative	Liikenteessä ei sallittu paketti, joka ei aiheuta hälytystä.
False Positive	Suomeksi ”väärä hälytys”. Paketti aiheuttaa hälytyksen, vaikka se kuuluu sallittujen pakettien joukkoon.
FIFO	First In, First Out. Jonon käsittely tapa, jossa ensiksi tullut tapahtumapyyntö käsitellään aina ensimmäisenä, toinen tullut toisena ja niin edelleen.
Floodlight	Avoimeen lähdekoodiin perustuva OpenFlow kontrolleri
GitHub	Avoimia lähdekoodia säilyttävä ja hallitseva palvelin.
GRE	Generic Routing Encapsulation on Ciscon kehittämä Ip-tunnelointi-protokolla.
ICMP	Internet Control Message Protocol, jota käytetään yhteyksien testaamiseen ja virheiden havainnointiin.
Java	Ohjelmistoalusta, jolla on oma oliopohjainen ohjelmointikieli.

JVM	Java Virtual Machine. Java pohjainen virtuaalikone
Kontrollitaso	Kontrollitaso (Data Plane) hallitsee laitetta päättämällä, kuinka paketteja käsitellään sekä täydentämällä MAC ja IP-tauluja.
LACP	Link Aggregation Control Protocol. LACP on protokolla, jota käytetään kun halutaan yhdistää kaksi tai useampaa verkkolinkkiä toisiinsa toimimaan yhtenä loogisena linkkinä.
Libcap-kirjasto	Kirjasto, jonka avulla esimerkiksi Snort ja Tcpdump analysoivat paketteja
MAC	Media Access Control. Laitteen verkkosovittimen yksilöllinen osoite Ethernet-verkossa
Man in the Middle	“Mies välissä-hyökkäys” on tietoturvahyökkäys, jossa ulkopuolinen taho piiloutuu kahden osapuolen välille salakuuntelemaan liikennettä.
MD5-enkryptaus	Message digest-algoritmi, jota käytetään varmistamaan tietoliikennepaketin eheys ja koskemattomuus.
Mininet	Ohjelma, jolla voidaan luoda virtuaalisia lähiverkkoja
MPLS	Multiprotocol Label Switching. menetelmä, jolla välitetään IP-paketteja runkoverkossa.
Multi-tenant	Verkkoratkaisu, jossa loppukäyttäjät erotetaan loogisesti toisistaan erilaisilla käytänteillä (VLAN, ACL).
MySQL	Tietokantaohjelmisto
NETCONF	Network Configuration Protocol. Verkonlaitteiden hallintaprotokolla.
NetFlow	Cisco-valmistajan verkon monitorointi protokolla
Northbound API	SDN-verkkotekniikassa tätä termiä käytetään SDN-kontrollerin ja SDN-aplikaatioiden välisestä kommunikoinnista.
OF-Config	OpenFlow Configuration and Management Protocol. Verkonhallintaprotokolla
ONF	Open Network Foundation on organisaatio, jonka tarkoitus on kehittää ja luoda SDN-verkkotekniikan standardeja.
OpenDaylight	SDN-verkon kontrolleri
OpenFlow	SDN-verkon OpenFlow-kytkimien hallintaprotokolla
OpenFlow-kytkin	SDN-verkon kytkin, joka kykenee käsittelemään OpenFlow-protokollaa.

OpenStack	Avoimeen lähdekoodiin pohjautuva ohjelmistoalusta, jolla voidaan luoda pilvipalveluja.
Output-komponentti	Snort-ohjelman komponentti, joka hoitaa analysoinnin tulosten luomisen.
OVS	Open vSwitch. Ohjelmisto, jolla luodaan virtuaalisia kytkimiä.
OVSDB	Open vSwitch Database on SDN-verkon hallintaprotokolla
Packet aggregation	Pakettien lomitusta
Packet Decoder	Snort-ohjelman komponentti, joka tutkii ja analysoi vastaanottamiensa pakettien sisältöä-
Packet sampling	Tietoliikennepakettien näyttenotto
Preprocessor	Snort-ohjelman komponentti, joka käsittelee dataa ennen Detection Enginelle siirtämistä
Promiscuous	Verkkorajapinnan tila, jossa se kaappaa kaikki havaitsemansa paketit
Protokolla	Yhetykskäytäntö, joka määrittelee, kuinka laitteet ja sovellukset kommunikoivat keskenään.
RPC	Remote procedure call. Luodaan asiakas/palvelin-tyyppinen yhteys.
Ryu	SDN-verkon kontrolleri
RyuApp	RyuApp hallinnoi Ryun ohjelmia.
Ryu-manager	Ryu-ohjelmien pääsuorittaja.
SDDC	Software-Defined data center. Virtuaalinen datakeskus
SDN	Software Defined Network.
SDN-aplikaatio	Ohjelma, joka suoritetaan SDN-kontrollerilla, ja jolla luodaan verkkolaitteita sekä verkon ominaisuuksia.
SDN-kontrolleri	SDN-verkkoa hallitseva laite
SDN-verkkolaite	SDN-verkon laite, jota SDN-kontrolleri ohjaa.
sFlow	Verkon monitorointi-protokolla.
Southbound API	SDN-verkkotekniikassa tätä termiä käytetään SDN-kontrollerin ja SDN-verkkolaitteiden välisestä kommunikoinnista.
Tcpdump	Komentokehoteelta ajettava pakettien analysointi ohjelma
TLS/SSL	Transport Layer Security / Secure Sockets Layer. Salausprotokolla.
Unified2	Snort-ohjelman lisäosa, jonka avulla dataa voidaan kirjata nopeasti binäärimuotoiseksi.
URL-osoite	Osoite, joka kertoo tietyn tiedoston sijainnin.

Välitystaso	Välitystaso (Forward Plane) välittää paketteja saapuvasta portista lähtevään porttiin välitystaulun mukaisesti.
Verkkosilta	eng. Bridge. Ohjelmisto tai laite, jonka avulla voidaan yhdistää kaksi tai useampi verkko toisiinsa. Usein käytetään synonyyminä sanalle verkkokytkin.
Virtuaalikone	Käyttöjärjestelmä, jonka sovellukset ja resurssit, on sijoitettu isäntäkoneeseen. Virtuaalikone käyttää isäntäkoneensa resursseja toimiakseen. Sovelluksien ja loppukäyttäjä kuitenkin näkevät pelkän virtuaalikoneen.
Virtualisointi	Virtualisoinnilla tarkoitetaan jonkin fyysisen laitteen sijoittamista fyysisen sijainnin sijaan loogiseen. Fyysisen resurssin piirteet ovat piilossa muilta sovelluksilta ja loppukäyttäjiltä.
VLAN	Virtual LAN eli virtuaalinen lähiverkko.
Vuosääntö	SDN-kontrolleri määrittää OpenFlow-kytkimelle vuosääntöjä, joiden mukaan kytkin käsittelee sille saapuvat paketit.
Vuotaulu	SDN-kontrolleri lähettää OpenFlow-kytkimelle vuosääntöjä, jotka kytkin tallettaa yhteen tai useampaan vuotauluunsa.
xFlow	Yhteisnimitys sFlow ja NetFlow-protokollista, joita molempia käytetään verkon monitorointiin.

1 Johdanto

Nykyinen verkkotekniikkaratkaisu on käynyt kömpelöksi ja mukautuu huonosti nykypäivän tarpeisiin. Jo viime vuosituhannella on alettu suunnittelemaan erilaista verkkotekniikkaa, joka mukautuu ja elää paremmin muuttuvan verkon mukana. Vasta nyt, kun verkot ovat laajoja ja monimutkaisia, on tämä Software Defined Network-verkkotekniikka alkanut herättämään kiinnostusta.

Tämän opinnäytetyön toimeksiantaja, Cyber Trust, haluaa tutkia SDN-verkkotekniikan mahdollisuuksia. Jyväskylän ammattikorkeakoulu toimii yhtenä Cyber Trust -ohjelman tutkimusosapuolena. DIGILE on yhteistyökumppaniensa kanssa luonut ohjelman, Cybert Trustin, jonka tavoite on palauttaa yksityisyys ja luottamus digitaaliseen maailmaan. He haluavat yhdessä tuoda tietoturvallisuuden helposti saatettavaksi jokapäiväiseen elämään, sekä tuoda suomalaisen tietoturva-teollisuuden maailmanmarkkinoiden kärkeen. (Kuosmanen 2015.)

Toimeksiantaja on kiinnostunut SDN-tekniikan uudenaikaisista ominaisuuksista, ja sen paremmasta mukautuvuudesta nykypäivän tarpeisiin. Jotta Cyber Trust voisi tarjota mahdollisimman laajan tutkimustuloksen, on se halunnut tutkia useampaa eri ratkaisua SDN-verkkokontrollerin suhteen. Siksi tässä työssä käytetään vähemmän tunnettua verkkokontrolleria, Ryu. Toimeksiantaja toivoi myös panostusta SDN-verkon tietoturvaan, joten pohditaan myös SDN-verkon tietoturva-asioita ja otetaan testiin yksi Ryu-kontrollerin integraatioista, Snort. Snort on avoimeen lähdekoodiin pohjautuva tietoliikenteen analysointiohjelmisto. Sen avulla voidaan etsiä haavoittuvuuksia tietoverkosta, paikantaa verkon tietoturva-aukkoja sekä dynaamisesti suojella tietoverkkoa tunkeilijoilta. Snort-ohjelmistolla on erityisen kattava IDS eli tunkeilijan estojärjestelmä, jonka toiminnollisuuksia ja soveltuvuutta asiakkaan SDN-verkkoon erityisesti tutkitaan.

1.1 Työn tavoitteet

Työn tavoitteena on edistää asiakkaan SDN-verkon tietoturvallisuutta tuomalla sinne tunkeilijan estojärjestelmä. Järjestelmä toimii niin, että Snort antaa hälytyksen Ryu-kontrollerille, kun se havaitsee poikkeuksellista, mahdollisesti haitallista liikennettä

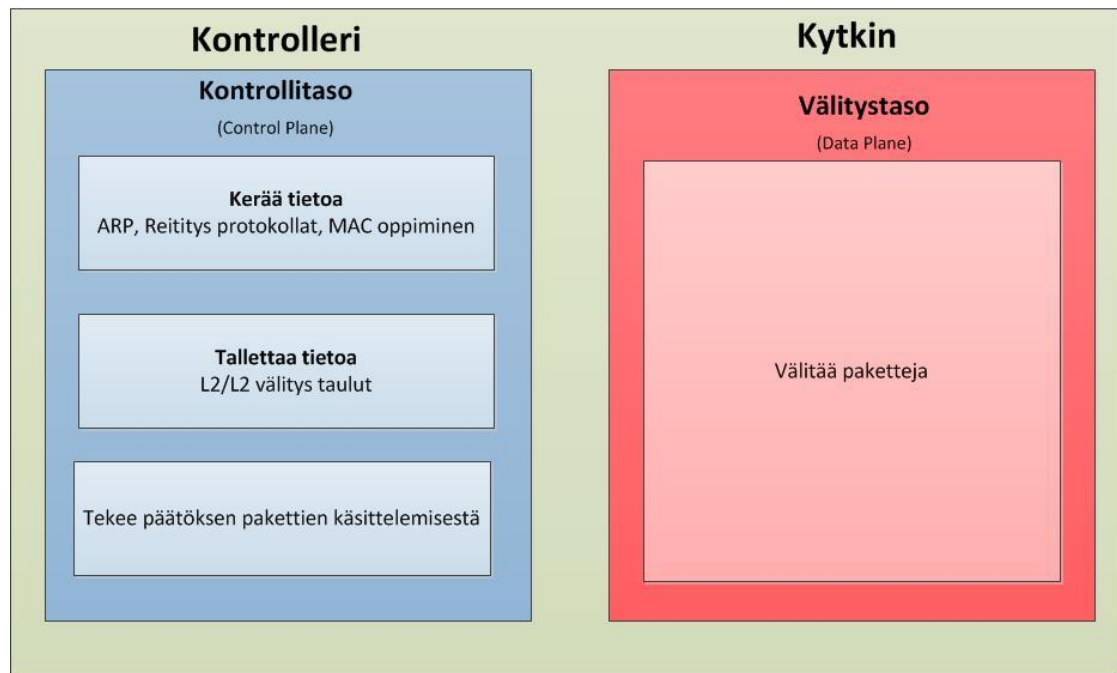
tietoverkossa. Ryu tekee hälytyksen perusteella ratkaisun. Esimerkiksi muokkaamalla verkonlaitteiden asetuksia niin, että hyökkäys keskeytyy, eikä uusia vastaavanlaisia hyökkäyksiä tietoverkkoon enää pääse.

Työn pohjana käytetään jo valmiiksi olemassa olevia Ryun applikaatioita, joiden toimivuutta testataan ensiksi hyvin alkeellisin testein edeten yhä haastavimpiin, ja oikeata tietoverkkoa suojaaviin menetelmiin. Työn tarkoituksena on selvittää, onko Snortilla sijaa SDN-verkon tietoturvassa, ja kuinka hyvin se pystyy vastaamaan SDN-verkon vaatimuksiin.

2 SDN-verkkotekniikka

2.1 SDN-verkon arkkitehtuuri

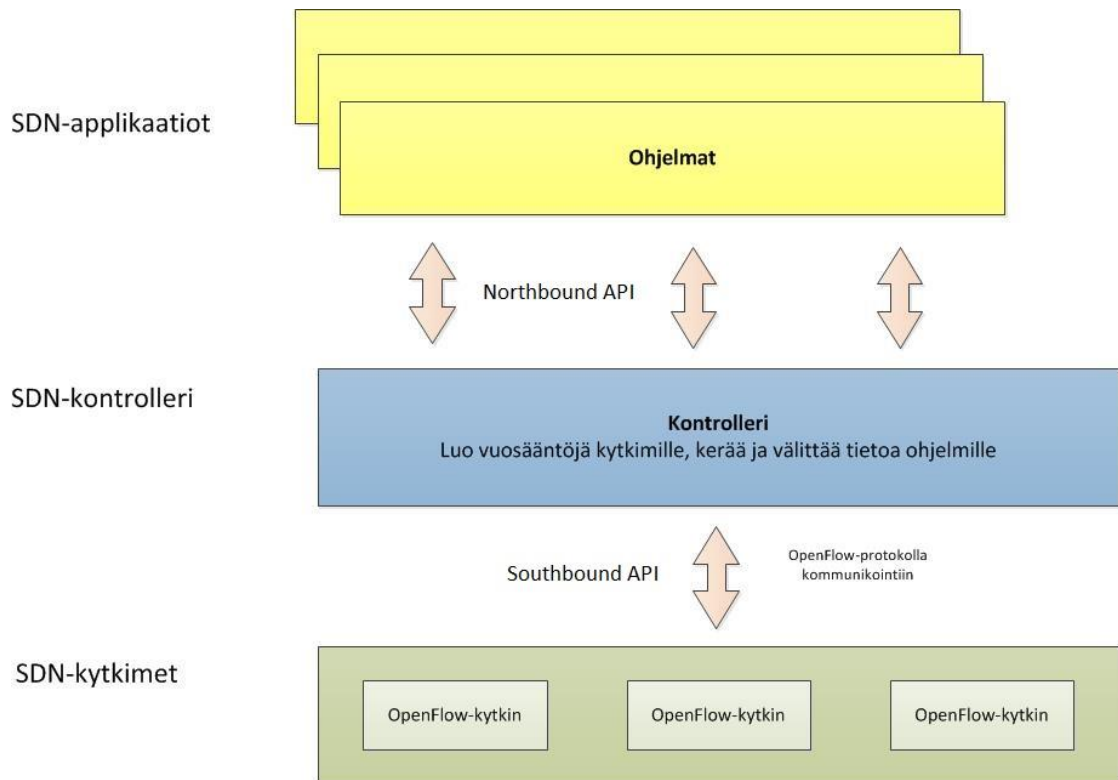
SDN-verkkotekniikalla on useampi määritelmä, mutta yleisin ja helpoin määritelmä on niin kutsuttu kontrolli- ja välitystason eriytyminen. Perinteisessä tietoverkossa jokaisella laitteella on oma kontrolli- ja välitystaso. Kontrollitason tehtävä on kerätä tietoa ympäröivästä verkosta, ja saadun tiedon perusteella tehdään päätös, kuinka saapuneita paketteja käsitellään. Välitystaso puolestaan nimensä mukaisesti hoitaa paketien välityksen kontrollitason keräämien ja tallennettujen tietojen perusteella. Kuvio 1 havainnollistaa, kuinka tasot on SDN-verkkotekniikassa erotettu toisistaan erillisille laitteille. Kontrollitaso on keskitetty yhdelle tai useammalle kontrollerille, jotka tekevät yksin päätöksen, kuinka verkonlaitteet käsittelevät niille saapuvat paketit. Täten, kun hallinnoija tekee verkkoon muutoksia, ei hänen tarvitse asentaa muutosta jokaiselle verkonlaitteelle. Hän tekee muutoksen ainoastaan kontrollerille, joka sitten välittää automaattisesti tiedon muille verkonlaitteille. Voidaan siis ajatella, että kontrolleri on verkon aivot ja muut laitteet ovat lihaksia, jotka tekevät itse työn. (Hedlug 2011.)



Kuvio 1. Kontrollitason ja välitystason eriytyminen (Hedlug 2011, Muokattu)

Ideana SDN-verkkotekniikka on yksinkertainen, mutta pinnan alla piilee paljon erilaisia protokollia ja ohjelmia. SDN:n voidaan ajatella koostuvan kolmesta elementistä; SDN-applikaatiot, SDN-kontrolleri ja SDN-verkkolaitteet (katso Kuvio 2). SDN-applikaatiot ovat ohjelmia, jotka kommunikoivat SDN-kontrollerin kanssa API (Application Programming Interface) rajapinnan kautta. Ohjelmien ja kontrollerin välistä rajapintaa kutsutaan Northbound-ohjelmointirajapinnaksi, kun taas kontrollerin ja verkkolaitteiden välistä rajapintaa kutsutaan Southbound-ohjelmointirajapinnaksi. Ohjelmien avulla voidaan hallita ja analysoida verkkoa, sekä korvata erinäisiä laitteita, kuten palomureja.

SDN-kontrolleri on puolestaan looginen osa, joka SDN-ohjelman pohjalta luo sääntöjä muille verkkolaitteille. Kontrolleri myös kerää tietoja verkkolaitteista ja välittää niitä takaisin SDN-ohjelmalle. Verkkolaitteiden tehtäväksi jää ainoastaan käsitellä paketteja kontrollerilta saamiensa sääntöjen mukaisesti. Jotta kytkimet ja kontrolleri voivat kommunikoida keskenään, tarvitaan erillinen protokolla. Yleisin ja tunnetuin protokolla on OpenFlow. (Inside SDN Architecture n.d.)



Kuvio 2. SDN-verkkotekniikan kolme elementtiä (What Is SDN 2013, Muokattu)

2.2 Virtualisointi tehostaa resurssien käyttöä

SDN-verkkotekniikassa käytetään pääasiassa virtuaalisia koneita, koska ne ovat paljon muuntautumiskykyisimpiä ja helpommin siirrettävissä, kuin fyysiset laitteet. Virtuaaliset koneet keskitetään suuriin data keskuksiin (SDDC, Software Defined Data Center). SDDC:n suurimpia hyötyjä muunneltavuuden lisäksi on se, ettei asiakkaan tarvitse rakentaa omaa verkkoinfrastruktuuriaan, vaan he voivat "vuokrata" laitteita palveluntarjoajaltaan. Tällöin palveluntarjoaja ei joudu tekemään minkäänlaisia fyysisiä muutoksia verkkoonsa. Kun verkon hallinta on keskitetty yhteen paikkaan, on verkon hallinta ja muokkaaminen helppoa. (What's a Software-Defined Data Center? n.d.)

Virtuaalisissa ympäristöissä käytetään virtuaalisia kytkimiä yhdistämään virtuaalisia koneita toisiinsa sekä muihin verkkoihin. Virtuaalisten kytkimien etu verrattuna tavalliseen fyysiseen kytkimeen on se, ettei virtuaalikoneita tarvitse siirtämisen ajaksi pysäyttää eikä asetuksia tarvitse määrittää uudelleen. Virtuaaliset kytkimet ovat myös

älykkäitä, sillä ne osaavat muuttaa virtuaalikoneen verkko- ja tietoturva asetuksia vastaamaan uuden ympäristön tietoturvakäytänteitä. Tällöin verkon tietoturvaan ei pääse syntymään tietoturva-aukkoja ihmiserheen vuoksi. (Black 2014.)

Yksi tunnetuin virtuaalikytkin-ohjelma on Open vSwitch. Se on avoimeen lähdekoodiin perustuva, ja tukee perinteisen kytkimen tavoin useimpia verkkoprotokollia (esimerkiksi NetFlow, sFlow, LACP, VLAN). Open vSwitch on suunniteltu erityisesti SDN-verkkoon, mutta se toimii myös perinteisessä verkkotekniikassa. OVSDB:tä (Open vSwitch Database) käytetään hallinnoimaan Open vSwitch-kytkintä. OVSDBn avulla voidaan lisätä, muokata ja poistaa verkkosilloja (eng. bridge) ja määrittää niiden asetuksia. (Black 2014.)

3 Ryu-kontrolleri

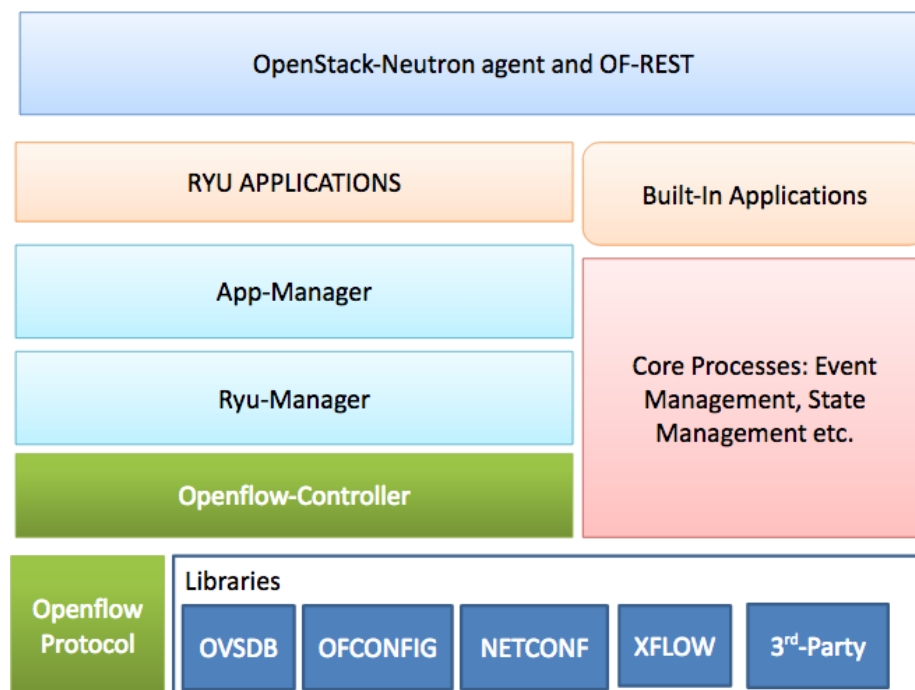
Ryu on japania ja tarkoittaa vuota (eng. flow). Ryu on SDN-kontrollerin viitekehys. Toisin sanoen, Ryu sisältää erilaisia komponentteja, joita muokkaamalla ja yhdistelemällä käyttäjä voi luoda erilaisia ohjelmia sopimaan juuri hänen käyttötarkoituksensa. Ryu perustuu avoimeen lähdekoodiin, ja mitä tahansa sen koodin osaa voidaan muokata. Ryu lähdekoodi on saatavilla Github-palvelimelta, jota ylläpitää ja kehittää Ryu-yhteisö. Myös OpenStack, pilvipalvelujen tarjoaja, tukee Ryun kehitystä. Usein Ryun kanssa käytetään OpenFlow-protokollaa, ja sen toimivuus on sertifioitu monella OpenFlow-kytkimellä, kuten Open vSwitchillä sekä Centec, Hewlett Packard ja IBM-valmistajien laitteilla. (What is Ryu Controller? n.d.)

3.1 Ryun arkkitehtuuri

Ryun vahvuutena on sen monipuolisuus. Ryu sisältää laajan kirjon erilaisia kirjastoja (Kuvio 3: Libraries), joiden avulla se pystyy käsittelemään erilaisia protokollia, kuten VLAN, MPLS ja GRE-tunnelointi protokollaa. OpenFlown lisäksi se tukee OF-Configia, OVSDB:tä, NETCONFia ja xFlowta. Tämän vuoksi Ryu sopii moneen erilaiseen verkkoratkaisuun. Esimerkiksi Netflow ja sFlow tukevat tietoliikennepakettien näytteenot-

toa (eng. sampling) sekä yhdistelemistä (eng. aggregation), joita usein käytetään verkon kaistankäytön mittauksissa. (Ryu, a Rich-Featured Open Source SDN Controller Supported by NTT Labs 2015.)

Ryun pääsuorittaja on Ryu-manager. Kaikki Ryun ohjelmat suoritetaan Ryu-managerin RyuApp luokan alla. Ryu-managerin avulla verkon OpenFlow-kytkimet voivat kommunikoida Ryun, sekä sen ohjelmien kanssa. Ryu sisältää useita valmiita ohjelmia, joilla saadaan aikaan verkon OpenFlow-kytkimillä erilaisia toiminnollisuuksia, kuten verkkokytkin, reititin ja palomuuuri. Lisäksi voidaan luoda VLANja ja GRE-tunneleita, sekä eristää verkon osia toisistaan. Ryu sisältää valmiiksi myös verkon havainnointityökalu (Topology Viewer), joka helpottaa käyttäjää tunnistamaan ja hahmottamaan verkonlaitteita graafisena piirroksena. Ryun arkkitehtuurin tärkeimpänä osana voidaan pitää OpenFlow-kontrolleria, jonka avulla Ryu hallinnoi OpenFlow-kytkimien vuosäntöjä. (SDN Series Part Four: Ryu, a Rich-Featured Open Source SDN Controller Supported by NTT Labs 2015.)



Kuvio 3. Ryun arkkitehtuuri (Sridhar 2015)

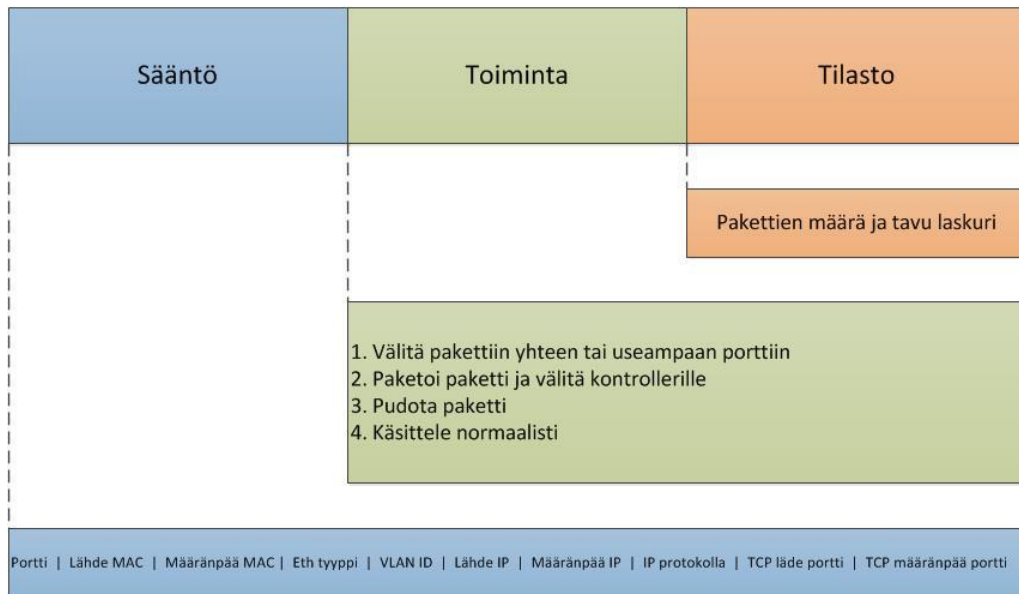
3.2 OpenFlow

OpenFlow protokolla on ensimmäisiä SDN-verkkotekniikan standardeja. Ryu tukee OpenFlow protokollaa aina versioon 1.4 saakka. Yleisin käytetty versio tällä hetkellä on kuitenkin vielä versio 1.3.

Vuo (eng. Flow) tarkoittaa paketteja, jotka kuljetetaan yhdeltä verkon päätelaitteelta toiselle. Yhdellä vuolla on aina tietty lähettäjä ja vastaanottaja. Täten voidaan ajatella, että yksittäinen vuo alkaa yhdestä pisteestä ja päättyy toiseen, kuten joki. Päätelaitteet tunnistetaan toisistaan muun muassa IP-osoitteen, TCP/UDP-portin ja VLAN-leiman avulla. (Black 2014). OpenFlow perustuu vuosäntöihin. SDN-verkon kontrolleri luo vuosäntöjä OpenFlow-kytkimille, jotka tallentuvat kytkimen vuotauluun (eng. flow table).

Kuvio 4 esittää vuotaulun toimintaperiaatteen. Kun kytkin vastaanottaa paketin, se etsii vuotaulusta pakettia vastaavan rivin. Vastaavuuden määrittämiseksi voidaan käyttää yhtä tai useampaan tietokenttää, kuten esimerkiksi lähettäjän tai vastaanottajan IP-osoitetietoja, MAC-osoitetietoja, VLAN-leimaa tai käytettyä protokollaa. Vastaavuutta verrattaessa paketin pitää täyttää kaikki säännössä määritetyt ehdot. Kun täydellinen vastaavuus löytyy, käsitellään paketti vuosäännön mukaisesti. Paketti voidaan esimerkiksi välittää edelleen, sitä voidaan muokata tai se voidaan pudottaa.

Vuotaulu



Kuvio 4. Vuotaulun toimintaperiaate (What is OpenFlow? n.d, Muokattu)

Jos saapuneelle paketille ei löydy sääntöä vuotaulusta, paketti välitetään kontrollerialle (OFPPacketIn-viesti, katso Kuvio 5), joka joko pudottaa paketin tai luo kytkimelle vuosäännön tulevaisuuden varalle (OFPPacketOut-viesti). Kontrollerin toiminta riippuu käytetystä OpenFlow-versiosta. Vuotauluun tallentuu myös tilastotietoa sääntöä vastaavasta paketista. (What is OpenFlow n.d.)

RYU OpenFlow Message

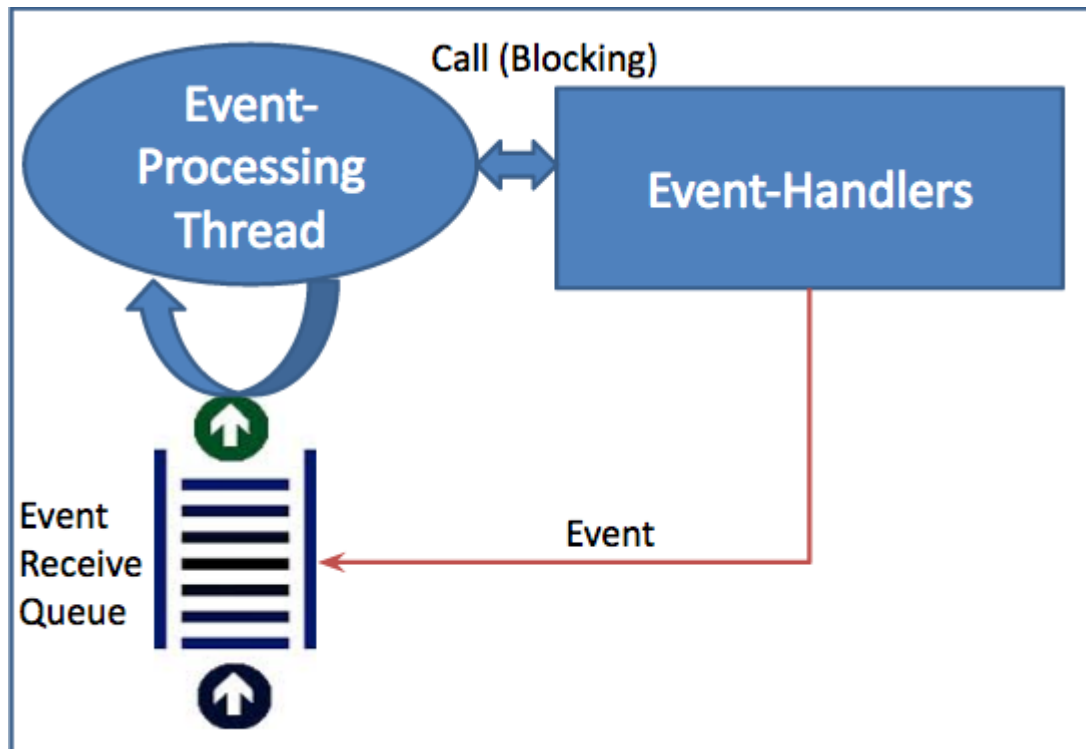
Type	Message Name	Ryu OpenFlow Message
Controller-to-Switch	Features	OFPPortStatsRequest / OFPSetConfig
	Configuration	OFPPortStatsReply
	Packet-out	OFPPacketOut
	Modify-State	OFPFlowMod
	Read-State	OFPFlowStatsRequest/OFPFlowStatsReply OFPPortStatsRequest/OFPPortStatsReply
	Barrier	OFPBarrierRequest/OFPBarrierReply
Asynchronous (switch->controller)	Packet-In	OFPPacketIn
	Flow Removed	OFPFlowRemoved
	Port Status	OFPPortStatus
	Error	OFPErrorMsg
Symmetric (switch<->controller)	Hello	OFPHello
	Echo Request / Reply	OFPEchoRequest/OFPEchoReply
	Vender	OFPExperimenter

Kuvio 5. Ryun OpenFlow-viestit (SDN Series Part Four: Ryu, a Rich-Featured Open Source SDN Controller Supported by NTT Labs 2015)

Ryu sisältää erillisen kirjaston, jonka avulla Ryu pystyy luomaan ja purkamaan OpenFlow-paketteja. Kuvio 5 voidaan nähdä minkälaisia erilaisia viestejä Ryu ja OpenFlow-kytkimet lähettävät toisilleen. Suurin osa viestejä ovat pelkästään Ryulta kytkimelle (Control-to-Switch), mutta verkon kytkimet käyttävät myös asynkronoituja viestejä kommunikoidakseen kontrollerin kanssa. Ne ilmoittavat näillä viesteillä kontrollerille verkon muutoksista ja lähettävät kontrollerille paketteja, joille niillä ei ole vuotauluissa vastaavaa sääntöä. Symmetrisiä (Symmetric) viestejä lähetetään niin kontrollerin, kuin verkon kytkimen toimesta. Ne ovat muun muassa yhteyden muodostamiseen käytettäviä viestejä.

3.3 Ryun toiminta perustuu tapahtumiin

Ryu ohjelmat on kirjoitettu Python-ohjelmointikielellä. Kuvio 6 havainnollistaa Ryun ohjelmien arkkitehtuuria. Jokaisella Ryun ohjelmalla on oma jono (Event receive queue), johon ohjelmalle saapuvat tapahtumat (Events) tulevat.



Kuvio 6. Ryun ohjelmien arkkitehtuuri (SDN Series Part Four: Ryu, a Rich-Featured Open Source SDN Controller Supported by NTT Labs 2015)

Tapahtumat ovat lähtöisin joko Ryulta itseltään tai muiden ohjelmien tapahtumankäsittelijöiltä (Event-Handlers). Tapahtumia käytetään ohjelmien väliseen kommunikointiin. Tapahtuma tyyppin perusteella ohjelma kutsuu tapahtumakäsittelijän, ja tapahtumat käsitellään saapumisjärjestyksessä (FIFO - First In First Out). Tapahtumia käsitellään yksi kerrallaan, ja tästä syntyykin nimitys yksisäikeinen ohjelma (single thread). Tapahtuma tulee aina suorittaa loppuun saakka, ennen kuin uutta tapahtumaa voidaan alkaa käsittelemään. (SDN Series Part Four: Ryu, a Rich-Featured Open Source SDN Controller Supported by NTT Labs 2015.)

3.4 Kontrollerien vertailua

OpenDaylight (ODL) on yksi suosituimmista SDN-kontrollereista. Sen suosio perustuu sen monipuolisuuteen ja graafiseen käyttöliittymään, joka useammilta, kuten Ryu-kontrollerilta, puuttuu. ODL on Linux Foundationin luoma, ja se on kehitetty Beacon-kontrollerin pohjalta. Sillä on useampia julkaisuja, joista viimeisin on Lithium. ODL on

Java-pohjainen, ja se ajetaan usein omalla Java-pohjaisella virtuaalikoneellaan (JVM – Java Virtual Machine). Ryun tavoin se on avoimeen lähdekoodiin perustuva sekä komponenttipohjainen. (Sridhar 2015.)

SDN-kontrollerien vertailu ei ole yksiselitteistä. Parhaan kontrollerin valintaan perustuu käyttötarkoitukseen sekä verkon rakenteeseen. Kun kontrollereja vertaillaan, huomioon otetaan kontrollerin tehokkuus, tuetut ominaisuudet, sekä niiden soveltuminen erilaisiin toteutuksiin. (Sridhar 2015.)

Kontrollerien **tehokkuutta** arvioidessa otetaan huomioon kontrollerin skaalautuvuus, luotettavuus, turvallisuus sekä suorituskyky. Suorituskykyyn vaikuttavat muun muassa tapahtumien käsittely nopeus, vastaus viive sekä tuettujen rajapintojen määrä. Kontrollerin tehokkuutta mitattaessa parhaimmiksi on arvioitu ODL, Ryu sekä Floodlight. (Sridhar 2015.)

Ominaisuuksien verrattaessa annettiin pisteytyksiä ominaisuudelle. Esimerkiksi TLS/SSH tuki ja virtualisoinnin mahdollisuus saivat suuremman pisteytyksen kuin dokumentoinnin määrä tai kontrollerin ikä. Vaikka OpenDaylight tukee useampia erilaisia ominaisuuksia kuin Ryu, ja sitä voi ohjata graafisen käyttöliittymän kautta, on Ryu valittu ominaisuuksiltaan kuitenkin parhaimmaksi kontrolleriksi. Valinta perustuu juurikin ominaisuuksien pisteyttämiseen. (Sridhar 2015.)

Vertailussa otettiin myös huomioon myös, kuinka kontrollerit soveltuivat **erilaisiin käyttötarkoituksiin**. Tällaisia sovellutuksia ovat esimerkiksi laitteen tuki erilaisille verkkolaitteille (kytkin, reititin), joilla voidaan luoda täydellinen yhteys eri verkoille. Tunnelointi, VLAN, OpenStack Neutron-lisäosa, L4-L7 -tason verkkotoiminnot, verkon monitorointi sekä käyttömahdollisuus kampusverkossa ovat vain muutamia eri esimerkkejä kontrollerin mahdollisista käyttötavoista. Alla oleva taulukko (Taulukko 1) tiivistää vertailun tuotoksen. 'Partial' merkitsee osittaista soveltumista kyseiseen käyttötarkoitukseen. Toisin sanoen kontrolleria voidaan käyttää kyseiseen tarkoitukseen, mutta siltä puuttuu huomattava määrä palveluita tai ohjelmia, jotta ominaisuutta voitaisiin käyttää helposti ja luotettavasti. ODL tukee lähes kaikkia verrattuja ominaisuuksia ja muutamaa vain osittain. (Sridhar 2015.)

Taulukko 1. Kontrollerin käyttötarkoitusten vertailu (Sridhar 2015)

Use-Cases \ Controllers	Trema	Nox/Pox	RYU	Floodlight	ODL	ONOS***
Network Virtualization by Virtual Overlays	YES	YES	YES	PARTIAL	YES	NO
Hop-by-hop Network Virtualization	NO	NO	NO	YES	YES	YES
OpenStack Neutron Support	NO	NO	YES	YES	YES	NO
Legacy Network Interoperability	NO	NO	NO	NO	YES	PARTIAL
Service Insertion and Chaining	NO	NO	PARTIAL	NO	YES	PARTIAL
Network Monitoring	PARTIAL	PARTIAL	YES	YES	YES	YES
Policy Enforcement	NO	NO	NO	PARTIAL	YES	PARTIAL
Load Balancing	NO	NO	NO	NO	YES	NO
Traffic Engineering	PARTIAL	PARTIAL	PARTIAL	PARTIAL	YES	PARTIAL
Dynamic Network Taps	NO	NO	YES	YES	YES	NO
Multi-Layer Network Optimization	NO	NO	NO	NO	PARTIAL	PARTIAL
Transport Networks - NV, Traffic-Rerouting, Interconnecting DCs, etc.	NO	NO	PARTIAL	NO	PARTIAL	PARTIAL
Campus Networks	PARTIAL	PARTIAL	PARTIAL	PARTIAL	PARTIAL	NO
Routing	YES	NO	YES	YES	YES	YES

4 Tietoturva

4.1 Tietoturva SDN-tekniikassa

Kun välitys- ja kontrolleritaso erotetaan ja verkon hallinta keskitetään yhteen paikkaan, on päivän selvää, että verkon kontrollerista tulee verkkohyökkäysten pääkohde. The Open Networking Foundation (ONF) kehittää käyttäjä-lähtöisesti avoimia standardeja SDN-tekniikkaan. (ONF Review. n.d.) Se on hyvin tietoinen tästä haavoittuvuudesta, ja onkin tunnistanut kaksi heikointa lenkkiä SDN-verkossa. Keskitetty kontrolleri on houkutteleva hyökkäyskohde, mutta sen lisäksi heikkona kohtana pidetään Southbound API -rajapintoja, kuten OpenFlowta. OpenFlowhin kohdistuva hyökkäys saattaa heikentää kontrollerin saatavuutta, suorituskykyä sekä verkon eheyttä. (McGillicuddy 2014.)

Kontrolleri on yleensä virtuaalikone, joka ajetaan isäntäkoneella. Isäntäkone on usein Linux-pohjainen käyttöjärjestelmä, joten sen tietoturva kannattaa ottaa myös huomioon. Isäntäkone kannattaa suojata käyttäjän tunnistamisella, salasanoilla sekä muilla tietoturva käytännöillä. South- ja Northbound-ohjelmointirajapinnoissa saattaa olla

myös käytössä paljon erilaisia protokollia, joissa on erilaisia haavoittuvuuksia. On tärkeää käyttää tietoturvallisia protokollia, jotka tukevat esimerkiksi TLS/SSL salausta sekä MD5 tarkistusta. Hyökkääjän on helppo käyttää salaamatonta protokollaa hyväkseen, ja sen avulla ottaa verkon kontrolleri haltuunsa. (McGillicuddy 2014.)

Kontrolleriin saattaa kohdistua DoS -hyökkäyksiä (Denial of Service, katso kappale 4.3), jotka alentavat kontrollerin saatavuutta, ja saattavat katkaista sen yhteyden kokonaan muihin verkkolaitteisiin. Jos kontrolleriin onnistutaan hyökkäämään verkonvalvojan huomaamatta, voi hyökkääjä määrittää kontrolleriin uusia vuosääntöjä, ja näin ohjata liikennettä toisaalle. Siten hyökkääjä voi rauhassa tutkia ja analysoida verkon liikennettä, ja löytää lisää tietoturva-aukkoja (Man in the Middle-hyökkäys, katso kappale 4.3). Hyökkääjä saattaa myös luoda verkkoon kokonaan uuden kontrollerin, jolloin hän saa koko verkon hallintaansa. (SDN Security Attack Vectors and SDN Hardening 2014.)

4.2 Snort kasvattaa tietoverkon tietoturvaa

Palomuri on tärkein komponentti, kun pyritään estämään hyökkäyksiä tietoverkkoon. Palomuri on nimensä mukaisesti muuri, jossa on portteja. Näistä porteista vain asianmukaiset paketit saavat kulkea läpi. Verkon tietoturvan hallinnoijan pyrkii sulkemaan kaikki portit, joita ei tarvita tavanomaiseen liikennöintiin. Näin muurin jää mahdollisimman vähän avoimia kohtia. Kaikesta huolimatta, palomuurista pääsee joskus matoja ja tunkeilijoita läpi, jolloin tarvitaan erillinen työkalu, jolla havaitaan verkkoon päässeet hyökkäykset. Tässä apuun tulee Snort-tunkeilijan havaitsemis- ja estojärjestelmä. (Hayes 2004.)

Tietoverkkotunkeilijat käyttävät yleensä tietynlaista "allekirjoitusta", joita etsimällä Snort kykenee tunnistamaan tunkeilijan. Allekirjoitukset ovat yleensä bittikuvoita tai tietoliikennepakettien lippuja (eng. flag), jotka eivät kuulu tavanomaiseen liikenteeseen. Jotkut, kehittyneemmät tunkeilijat, osaavat piilottaa itsensä tavanomaiseen liikenteeseen sekaan ilman allekirjoituksia, jolloin heidän havaitseminen on vaikeaa. Snort sisältää valmiiksi asennettuna suuren määrän erilaisia sääntöjä, joilla se kykenee havaitsemaan tunkeilijat. Lisäksi Snortilla on päivitystoiminnon, jonka kautta voidaan

päivittää uusia sääntöjä järjestelmään sitä mukaan, kun uudenlaisia hyökkäystapoja tunnistetaan. (Hayes 2004.)

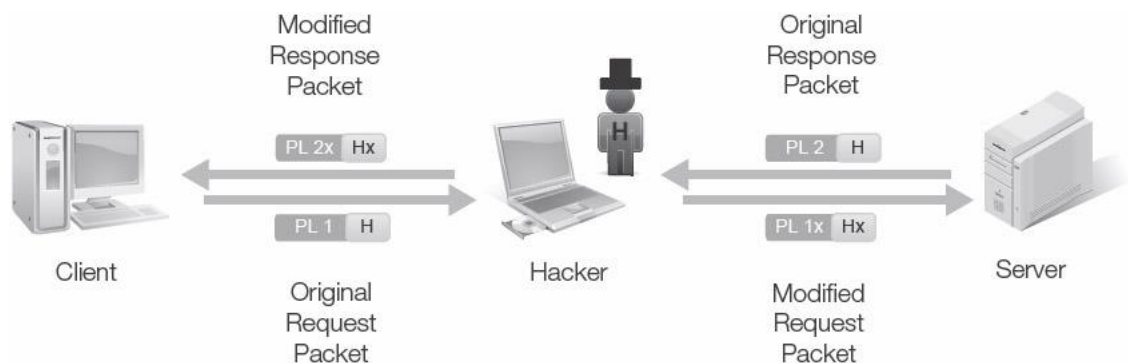
4.3 DoS ja Man in the Middle -hyökkäys

TCP-yhteys muodostetaan, kun asiakaslaite lähettää palvelimelle TCP-paketin, jossa on SYN-lippu. Palvelin vastaa siihen lähettämällä SYN-ACK-lipulla varustetun viestin takaisin asiakaslaitteelle, ja jää odottamaan ACK-lipulla varustettua viestiä takaisin. SYN-lippuja käyttämällä voidaan toteuttaa DoS-hyökkäys. Kuviossa 7 on esimerkki Syn Flood-hyökkäyksestä, jossa palvelimelle lähetetään kymmeniä tuhansia yhteyden avauspyyntöjä (SYN) kerralla. Palvelin vastaa avauspyyntöihin, ja jää odottamaan asiakaslaitteen kuittausta (ACK). Kuittausta ei kuitenkaan koskaan lähetetä takaisin, vaan sen sijaan lähetetään lisää yhteyden avauspyyntöjä. Tällöin palvelin voi mennä jumiin, eikä pysty vastaamaan enää oikeisiin yhteyden avauspyyntöihin. Tämä on vain yksi tapa toteuttaa DoS eli palvelunestohyökkäys.



Kuvio 7. SYN-flood hyökkäys (What Is TCP Syn Flood Attack 2015, Muokattu)

Joskus tunkeilija ei hyökkää suoraan aggressiivisesti kohteeseensa, vaan piiloutuu ja salakuuntelee verkkoa sen ulkopuolelta. Man in the middle-hyökkäys on yksi tavannomaisista hyökkäystavoista, jossa ulkopuolinen taho ujuttautuu kahden keskustele- van laitteen välille (Katso Kuvio 8). Hyökkäys toteutetaan niin, että tunkeilija antaa väärä tietoa asiakaslaitteelle lähettämällä väärennettyjä ARP, DNS tai ICMP-viestejä verkkoon. Näin tunkeilija huijaa asiakaslaitetta (eng. client) luulemaan, että tunkeilija on se palvelin, johon tämän on tarkoitus ottaa yhteyttä. Asiakaslaite muodostaa yhteyden tunkeilijan laitteeseen, ja tunkeilija välittää viestit edelleen palvelimelle, johon asiakaslaiteen alun perin piti ottaa yhteyttä. Tunkeilija välittää viestit palvelimen ja asiakaslaitteen välillä niin, ettei kumpikaan huomaa, että välissä on ulkopuolinen taho. Yhteyden salakuuntelu voi jatkua pitkään, ja tunkeilijan tarkoitus on yleensä saada käsiinsä arkaluonteista tietoa, kuten salasanoja, valtuutustietoja tai muuta salassa pidettävää aineistoa. (Al Braiki 2013.)

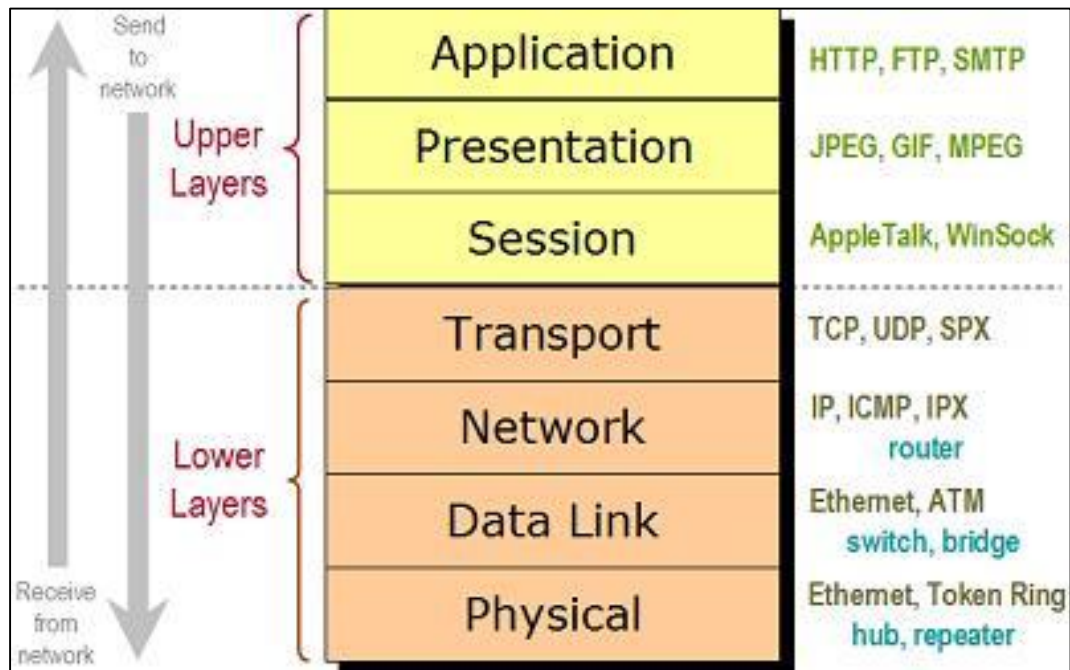


Kuvio 8. Man-in-the-middle hyökkäys, mies yhteyden välissä (Stewart 2014)

5 OSI-malli

Open Systems Interconnection (OSI) malli on alun perin suunniteltu tietoverkkoprotokollien kehittämiseen. Mallin avulla eri valmistajat voivat luoda omia protokollia ja varmistaa niiden yhteensopivuus muiden valmistajien protokollien kanssa. OSI-malli on jaettu seitsemään eri tasoon, jossa jokaisella tasolla on oma tehtävänsä. Tasot on

havainnollistettu Kuviossa 9. OSI-mallin hyöty on se, että sen avulla on voitu jakaa monimutkainen tietoliikennepaketin sisältö pienempiin osiin. Esimerkiksi, jos halutaan muuttaa vain tietoliikennepaketin IP-osoitetta, muutetaan vaan dataa kolmannella OSI-tasolla, jolloin vaikutusta muihin kerroksiin ei ole. (Ur Rehman 2003.)



Kuvio 9. OSI-malli (Mitchell 2015)

Snort keskittyy tutkimaan OSI-mallista erityisesti L3 ja L4-tasojen pakettien otsikkotietoja, mutta myös joidenkin L7-tason protokollien ohjelmatietoja, kuten esimerkiksi HTTP:tä. OSI-mallin rakenteen tunteminen on tärkeää, kun luodaan omia Snort sääntöjä. OSI-mallin tasot määritellään seuraavanlaisesti: (Thomas n.d.)

Ensimmäinen taso käsittää itse raudan eli fyysisen verkkokortin.

Toisella tasolla (L2) määritetään yhteyteen käytettävän median tyyppi. Medialla tarkoitetaan siirtokanavaa, jolla yhteys muodostetaan. Näitä ovat esimerkiksi kupari,

koaksiaalinen tai optinen johto. Tällä tasolla määritetään tarkkaan kuinka data siirretään fyysiselle medialle, kuinka bitti jono muutetaan elektroniseksi tai optiseksi signaaliksi sekä määritetään Ethernet-osoitteet.

Verkkotaso (L3) on vastuussa päätelaitteiden välisestä yhteydestä sekä datan eheydestä. Data jaetaan paketteihin ja niihin lisätään paketin lähettäjän ja vastaanottajan osoitteet (IP tai looginen osoite). Tällä tasolla paketit myös järjestetään oikeaan järjestykseen ennen seuraavalle tasolle siirtämistä.

Transport-tasolla (L4) muodostetaan yhteys vastaanottajan ja lähettäjän välille. Tätä tasoa hallitsee TCP ja UDP-protokollat. TCP-protokollan avulla luodaan luotettava, keskustelumuotoinen yhteys. Yhteyden alussa päätetään yhteyden säännöistä, sekä kuinka keskustelu laitteiden välillä etenee. Jokainen vastaanotettu paketti kuitataan. UDP-protokollalla puolestaan luodaan yhteys, jolla paketit lähetetään ilman kuitausta tai varmuutta paketin perille pääsemisestä.

Tasoilla L5-L7 luodaan palvelut, joita käytetään verkossa. Näitä ovat esimerkiksi DNS, HTTP, SSH, FTP ja DHCP. Useimmilla näistä on oma protokolla, jota käytetään palvelun luomiseen. Tasot koostuvat yhteyksien ylläpidosta, datan käsittelystä ja salauksesta sekä loppukäyttäjälle näytettävän datan toteuttamisesta.

6 Snort

Tietoverkkojen uhkana ovat erilaiset tietoturvamurrot. Murron takana voi olla mato, joka on päässyt palomuurista läpi tai hakkeri, joka on päässyt verkkojärjestelmään. Verkkoon voi kohdistua DoS-hyökkäys, tai verkon turvallisuutta uhkaa joku sisäverkosta nuuskimalla tietoja, jotka eivät hänelle kuulu. Tietoturvamurrosta puhutaan silloin kun tietoverkon turvallisuuteen vaikutetaan heikentävästi ja ne vaativat välittömiä toimenpiteitä. Tietoturvamurto voi olla kohtalokas verkolle. Verkon hyökkääjä voi saada koko verkon kaatumaan, tai kaapata sen itsellensä, jolloin koko yrityksen toiminta on vaakalaudalla. Kun verkossa havaitaan tunkeilija, on verkon hallinnoijilla täysi työ keskeyttää hyökkäys ja saada verkko takaisin haltuunsa. (Hayes 2004.)

6.1 Snortin ominaisuudet

Snort on tällä hetkellä yksi käytetyimmistä avoimeen lähdekoodiin perustuvista tunkeilijanjärjestelmistä. Snort on hyvin mukautuvainen erilaisiin ympäristöihin, ja sitä voidaan ajaa useimmissa käyttöjärjestelmissä ja -ympäristöissä. Lisäksi se saa jatkuvasti lisäpäivityksiä. Snort perustuu perinteiseen tcpdump-työkaluun, jolla analysoidaan pakettien sisältöjä. Tcpdump-työkalua kehittämällä ja hienosäätämällä on saatu tehokkaampi työkalu, jolla voidaan entistä tarkemmin tutkia ja tunnistaa tietoliikennepakettien sisältöä. (Hayes 2004.)

Snortin päätarkoitus on pysäyttää verkon sisälle pääseiden tunkeilijoiden eteneminen. Sen lisäksi se pitää sisällään työkaluja, joiden avulla voidaan etukäteen analysoida oman verkon mahdollisia tietoturva-aukkoja. Snortin avulla voidaan skannata verkon laitteiden avoimia portteja, ja arvioida palomuurin toimivuutta. Se kertoo, jos joku verkon sisällä rikkoo tietoturvakäytänteitä, esimerkiksi käyttämällä sovellusta, joka on luokiteltu mahdollisesti tietoturvaa heikentäväksi. Lisäksi, jos tunkeilija on päässyt verkkoon, se tarjoaa yksityiskohtaista tietoa hyökkääjästä, ja kertoo kuinka pitkälle hyökkääjä pääsi tunkeutumaan verkkoon. (Hayes 2004.)

Snort tukee kolmea tilaa, joilla se analysoi verkkoa. Sniffer- ja Packet Logger-tilat ovat tarkoitettu vain verkon tutkimiseen. Näissä tiloissa Snort ei itse tutki tai analysoi paketteja, vaan tietojen analysointi jää käyttäjälle.

Sniffer-tilassa Snort kirjaa jokaisen kaappaamansa paketin tiedot konsolille. Mitään ei siis talleteta, eikä paketeista tule hälytyksiä. Käyttäjä saa itse valita, kuinka paljon tietoa hän haluaa Snortin kaappaamista paketeista. Snort voi kirjata joko ainoastaan TCP/IP-pakettien otsikkotiedot, tai niiden lisäksi myös paketin sisällä olevan datan.

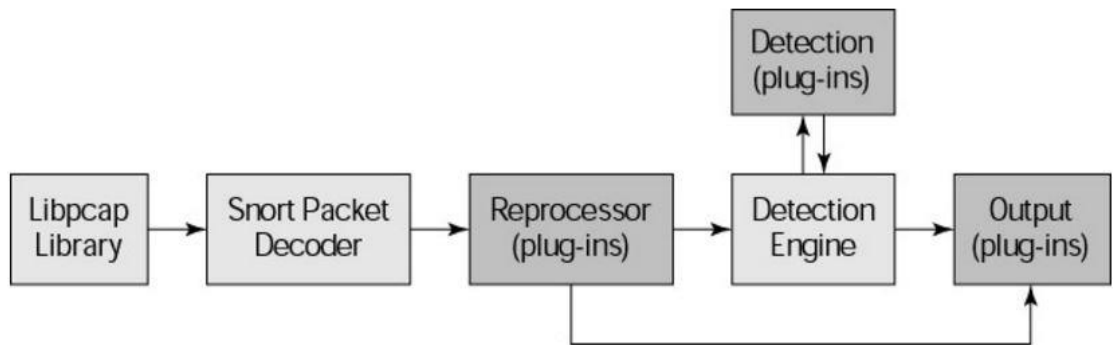
Packet Logger-tilassa Snort puolestaan tallettaa kaikkien kaappaamiensa pakettien tiedot lokiin. Sniffer-tilan tavoin, käyttäjä saa itse valita kuinka paljon hän haluaa tietoja tallentuvan, mutta myös minne tieto tallentuu. Näistä kahdesta vaihtoehdosta Logger-tila on hyödyllisempi, koska dataa voidaan myöhemmin analysoida rauhassa. Näitä kahta ominaisuutta voidaan käyttää hyödyksi, kun suunnitellaan Snortin IDS-tilan sääntöjä, tai muuten halutaan tutkia verkon liikennettä. (Snort manual 2015.)

Kolmantena tilana Snortilla on IDS, joka kaappaa paketteja medialta, tutkii ja analysoi niitä. On kaksi eri tapaa, jolla IDS voidaan ajaa. Sitä käytetään, joko tutkimaan koko verkon liikennettä (NIDS), tai sitten ainoastaan yhdelle laitteelle tulevaa liikennettä (HIDS). HIDS eli Host-based IDS tarkkaillee järjestelmän toimintaa ja sille tulevia istuntoja. Se hälyttää epäilyttävästä toiminnasta, kuten esimerkiksi useasta peräkkäisestä epäonnistuneesta kirjautumisyrityksestä. Se myös varmistaa sille tulevien pakettien eheyden, ettei ulkopuolinen taho ole päässyt muokkaamaan tai salakuuntelemaan paketteja. (Seagren 2008.)

6.2 Snortin komponentit

NIDS-tilassa Snortiin pystytään määrittämään, millaisia paketteja sen halutaan tutkivan. Suuressa verkossa kaikkien pakettien analysoiminen vie hurjasti resursseja. Siksi on yleensä parempi asettaa suodatin, jonka perusteella Snort valitsee tutkittavat paketit. Tällä hetkellä Snortiin voidaan määrittää vain yksi suodatin, jonka perusteella se kaappaa paketit medialta. Mikään ei kuitenkaan estä ajamasta useampaa Snort-instanssia yhtä aikaa. (Hayes 2004.)

Kuvio 10 havainnollistaa Snortin arkkitehtuuria. Snortin toiminnan mahdollistaa verkkokortin asettaminen niin kutsuttuun promiscuous-tilaan, jolloin verkkokortti pystyy kaappaamaan tietoliikenteestä myös paketit, joita ei ole osoitettu sille. Lisäksi Snort tarvitsee Lipcap-kirjaston, jonka avulla se kaappaa paketit verkkokortilta. Paketit ovat niin kutsuttua raakadataa, joten paketit tulee käsitellä, jotta Snort pystyy tutkimaan niiden sisältöä. Packet Decoderin tehtävä on ottaa L2-data (OSI-malli) Lipcap-kirjastolta, ja purkaa paketista esiin IP-protokolla sekä TCP/UDP-paketti. Packet Decoder välittää puretut tiedot eteenpäin Preprocessorin ja Detection Enginen käsiteltäväksi. Preprocessor tai Detection Engine analysoivat tietoa ja tekevät hälytyksiä annettujen sääntöjen perusteella. Hälytys välitetään Output-komponentille, joka joko kirjaa hälytyksen lokiin, konsolille, tietokantaan tai muuhun käyttäjän määrittelemään sijaintiin. (Hayes 2004.)



Kuvio 10. Snortin komponentit (Hayes 2004)

6.2.1 Preprocessors – Esikäsittelijät

Preprocessorit ovat kokoelma erilaisia lisäosia, jotka voi kytkeä päälle tai pois päältä tarpeen mukaan. Preprocessorin vaatima prosessoriteho on huomattavasti pienempi kuin Detection Enginen, joten niiden pääasiallinen tehtävä on vähentää prosessorille syntyvää kuormaa. Jos kaikki paketit menisivät suoraan Detection Enginelle käsiteltäväksi, prosessori ylikuormittuisi, eikä Snort pystyisi käsittelemään läheskään kaikkia sille tulevia paketteja. Tällöin tunkeilijan on mahdollista päästä livahtamaan Snortin ohi. (Hayes 2004.)

Preprocessorin tehtävä on myös muokata saamaansa dataa niin, että Detection Enginen on helpompi tulkita sitä. Kun verkon käyttäjä esimerkiksi selailee WEB-sivustoja, tai katsoo online-videoita, TCP-protokollan avulla muodostetaan keskustelumuooinen yhteys käyttäjän ja WEB-palvelimen välille. Tällä keskustellulla on hyvin selkeä ja ennalta määritetty rakenne, ja sitä kutsutaan istunnoksi. Istunnossa paketteja liikkuu palvelimelta käyttäjälle ja takaisin selkeässä kaavassa. Ilman Preprocessoria, nämä TCP-yhteyden paketit olisivat Detection Enginen silmissä toisistaan täysin irralliset, eikä Detection Engine pysty näkemään keskustelussa minkäänlaista kaavaa. Stream5-lisäosan avulla Snort pystyy kasaamaan eri TCP-paketeista istuntoja, jolloin Detection Engine saa selkeän kuvan, mitä pakettien tulisi pitää sisällään ja näin tunnistaa mahdollisia tunkeilijoita. (Alder 2004.)

Joskus myös TCP-keskustelun aikana, jotkin paketit saattavat kulkea eri reittiä kuin toiset paketit, jolloin ne saavuttavat määränpänsä hieman eri järjestyksessä. Preprocessor-lisäosa Frag3 järjestää saapuneet paketit oikeaan järjestykseen. Preprocessor saattaa joskus huomata jo pakettien järjestelemisen aikana outoja merkintöjä paketeissa, tai paketteja, jotka eivät liity mihinkään keskusteluun. Tällöin Preprocessor voi tehdä itse hälytyksen, ja näin säästää Detection Enginen resursseja. (Hayes 2004.)

Verkon laitteet ja verkon käyttäjät käyttävät usein eri kieliä kommunikoidakseen. Esimerkiksi URL-osoitteen voidaan kirjoittaa useammalla eri tavalla, vaikka ne tarkoittavatkin tismalleen samaa asiaa. Kun Snortiin tehdään sääntöjä, on usein vaikeaa ottaa huomioon kaikki mahdolliset kielet ja koodaustavat, jolla sama asia voidaan ilmaista. Snort lukee sääntöjä kirjaimellisesti, joten tunkeilijoille saattaa jäädä avoimia ovia kielimuurin vuoksi. Snortilla on niin kutsuttuja verkon normalisointi-lisäosia, joiden avulla se kääntää eritavalla kirjoitettuja tekstejä ”normaaliin”, käyttäjälle tutumpaan tapaan. Tällöin Snortin sääntöjä ei tarvitse kirjoittaa jokaisella mahdollisella kielellä. (Alder 2004.)

Kuviossa 11 oleva esimerkki havainnollistaa datan normalisointia käytännössä. Tässä tapauksessa hyökkääjä yrittää manipuloida ohjelman tiedostoa, ja siten päästäkseen käsiksi itse ohjelmaan. Koska hyökkääjä käyttää hyökkäyksessään HTTP-yhteyttä, URL-osoite kirjoitetaan hieman eri tavalla, kuin verkon hallinnoija on ajattelut. Vaikka verkon hallinnoija on tehnyt säännön, jonka tarkoitus on estää hyökkäyksiä kyseiseen tiedostoon, on sääntö turha. Sääntö ei laukaise hälytystä, koska hyökkäys ei täytä kirjaimellisesti säännön vaatimuksia. URL-normalisoijan avulla tunkeilijan HTTP-yhteydessä käytetty URL-osoitteen ulkoasu muuttuu vastaamaan hallinnoijan sääntöä, ja näin laukaisee hälytyksen. (Biles 2005.)

<pre> Hyökkääjän käyttämä URL-osoite: /scripts/..%c0%af../winnt/system32/cmd.exe?/c+ver Normaalisoinnin myötä muuttunut osoite: /winnt/svstem32/cmd.exe?/c+ver </pre>
--

Kuvio 11. Esimerkki liikenteen normalisoinnista (Biles 2005, Muokattu)

6.2.2 Detection Engine

Detection Engineä voidaan pitää Snortin sydämenä. Sen tehtävä on etsiä paketeista niin kutsuttuja allekirjoituksia, joiden perusteella se havaitsee tunkeilijan. Allekirjoituksia luodaan tunkeilijalle tyypillisestä toiminnasta. Koska tunkeilijoilla on aina tietynlaisia tapoja toteuttaa hyökkäyksensä, voidaan tunnistettujen hyökkäysten pohjalta luoda allekirjoituksia. Allekirjoituksia käytetään Snort-sääntöjen pohjana. Jokaisesta verkkokortilta kaapattua pakettia verrataan jokaiseen Snortin sääntöön, ja jos vastaavuus löytyy, toteutetaan säännön mukainen toiminto. (Sanders 2014.)

Detection Engine tutkii paketeista IP, TCP/UDP ja ohjelmatason otsikkotiedot sekä pakettien sisältämää dataa. Detection Engine vaatii huomattavasti resursseja järjestelmältä. Jos verkossa on paljon liikennettä, ei Detection Engine välttämättä ehdi analysoida kaikkia paketteja, jolloin se saattaa pudottaa osan paketeista. Yhdistelmällä ja vähentämällä Snortiin asetettuja sääntöjä voidaan vähentää Detection Enginelle syntyvää kuormaa. Sääntöjä luodessa tulee löytää kultainen keskitie, jotta saadaan mahdollisimman matala false negative-taso ilman järjestelmän liiallista kuormittamista. (Ur Rehman 2003.)

Vanhemmissa Snortin versioissa Snort keskeyttää pakettien tutkimisen, jos vastaavuus sääntöön löytyy, ja tekee hälytyksen säännön perusteella. Näin Snort säästää resurssejaan, mutta toiminnassa on yksi ongelma. Snortin sääntöjä voidaan luokitella eri prioriteetteihin, jolloin matalamman prioriteetin omaava sääntö ei aiheuta niin huomiota herättävää hälytystä, kuin korkeamman prioriteetin. Jos vanhempi Snort-versio löytää vastaavuuden matalan prioriteetin omaavaan sääntöön, paketin tutkiminen keskeytetään. Jos paketissa on toinen, korkeamman prioriteetin sääntöä vastaava tieto, jää se Snortilta huomaamatta. Tällöin välittömiä toimia vaativa tietoturvahyökkäys voi jäädä kokonaan huomaamatta, ja tunkeilija voi rauhassa jatkaa toimintaansa. Uudemmassa Snort-versiossa kaikkia sääntöjä verrataan tullessiin paketteihin ja hälytys toteutetaan korkeimman prioriteetin omaavan säännön mukaisesti. (Ur Rehman 2003.)

6.2.3 Output-moduuli

Snortin käyttäjä saa valita, kuinka Snort tallettaa tietoja hälytyksistä. Snort antaa vapauden valita tehdäänkö tunkeilijahavainnoista hälytys konsolille, vai kirjataanko tiedot ainoastaan lokiin myöhempää analysointia varten. Useimmiten käytetään kumpaakin vaihtoehtoa yhtä aikaa. (Alder 2004.)

Snortilla on IDS-moodissa seitsemän perustilaa, joista voidaan valita, kuinka se antaa hälytyksen mahdollisesta tunkeilijasta. Oletuksena Snort käyttää Full-hälytystoimintoa, jossa se kirjaa lyhyen hälytyksen konsolille, sekä tallettaa paketin täydelliset otoskottiedot loki-tiedostoon. Oletuksena lokimerkintä tehdään `/var/log/snort-` hakemistoon, jossa tiedot jaetaan IP-osoitteen perusteella eri kansiohakemistoihin. Oletussijainnin sijaan voidaan myös määrittää muita sijainteja, kuten järjestelmäloki (syslog) tai ulkoinen tiedostopalvelin. Tietojen tallettaminen tietokantaan on yksi Snortin suosituimmista talletusmuodoista. Tietokannassa tiedot pysyvät selkeässä järjestyksessä, ja sieltä ne on helppo jakaa muille. (Snort users manual 2015.)

Jos verkossa on paljon liikennettä, suositetaan käyttämään Fast-hälytystoimintoa, joka kirjaa tiedot huomattavasti nopeammin talteen kuin Full-hälytystoiminto. Fast-hälytystoiminnossa Snort tarvitsee vähemmän resursseja hakemistorakenteen luomiseen, tietojen purkamiseen sekä tallettamiseen. Unsock-hälytystoiminnolla Snort lähettää hälytyksen ulkopuoliselle UNIX-järjestelmälle. Tätä toimintoa tullaan tarkastelemaan myöhemmin tässä opinnäytetyössä. (Snort users manual 2015.)

Kuviossa 12 on esimerkki Fast-hälytyksestä. Siinä Snort kirjaa konsolille tiedot hälytyksen laukaisseesta NMAP-skannaustyökalusta. NMAP on järjestelmän skannaustyökalu, jolla tutkitaan verkon laitteita, niiden osoitteita ja avoimia portteja. Tällä työkalulla tunkeilija usein nuuskii tietoa verkosta ja sen mahdollisista murtokohdista. (Seagren 2008.)

```
01/20-22:34:35.218093  [**] [1:469:1] ICMP PING NMAP [**] [Classification:  
  Attempted Information Leak] [Priority: 2] {ICMP} 192.168.1.68 ->  
  172.16.34.18
```

Kuvio 12. Esimerkki Snort-hälytyksestä (Hayes 2004)

Kuvion 12 hälytys sisältää seuraavia tietoja:

- Päivä- ja aikaleima ilmoitettuna millisekunnin tarkkuudella.
- Allekirjoituksen tunnistustieto *
- Lyhyt selitys hälytyksestä; ICMP PING NMAP
- Hyökkäyksen luokittelu ja prioriteetti
- Protokolla, jota käytettiin hyökkäyksessä
- Paketin lähettäjän ja vastaanottajan IP-osoitteet

* Allekirjoituksen tunnistustiedon avulla voidaan paikantaa tarkkaan, mikä sääntö laukaisi hälytyksen. Ensimmäinen arvo on Generator ID (gid), joka määrittää mikä Snortin komponenteista aiheutti hälytyksen. Toinen arvo on Snortin säännön ID (SID) ja kolmas on säännön versio numero (rev). (Hayes 2004.)

Kuvio 13 on otettu samasta tilanteesta kuin Kuvio 12, mutta tässä tapauksessa tiedot on talletettu lokiin. Kuten kuviosta nähdään, tallentuu lokiin hieman enemmän tietoa paketista, kuin konsolille. Lokiin talletetaan IP-paketin otsikkotiedot. Kuvassa näkyy myös porttiskannaus porttiin 80, joka on hyvin tavanomainen NMAP - skannaus työkalun käytössä. (Hayes 2004.)

```

01/14-19:42:03.114656 0:10:67:0:B2:50 -> 0:A0:CC:D2:10:31 type:0x800 len:0x3C
192.168.1.68 -> 172.16.34.18 ICMP TTL:37 TOS:0x0 ID:44936 IpLen:20 DgmLen:28
Type:8 Code:0 ID:13988 Seq:7720 ECHO

+++++

01/14-19:42:03.114717 0:A0:CC:D2:10:31 -> 0:10:67:0:B2:50 type:0x800 len:0x2A
172.16.34.18 -> 192.168.1.68 ICMP TTL:255 TOS:0x0 ID:2734 IpLen:20 DgmLen:28
Type:0 Code:0 ID:13988 Seq:7720 ECHO REPLY

+++++

01/14-19:42:03.115157 0:10:67:0:B2:50 -> 0:A0:CC:D2:10:31 type:0x800 len:0x3C
192.168.1.68:50488 -> 172.16.34.18:80 TCP TTL:36 TOS:0x0 ID:3836 IpLen:20
DgmLen:40
***A*** Seq: 0x3C4A079E Ack: 0x498A079E Win: 0x800 TcpLen: 20

+++++

01/14-19:42:03.115194 0:A0:CC:D2:10:31 -> 0:10:67:0:B2:50 type:0x800 len:0x36
172.16.34.18:80 -> 192.168.1.68:50488 TCP TTL:255 TOS:0x0 ID:0 IpLen:20
DgmLen:40 DF
*****R** Seq: 0x498A079E Ack: 0x0 Win: 0x0 TcpLen: 20

+++++

```

Kuvio 13. Esimerkki Snortin loki-merkinnästä (Hayes 2004)

Seuraava esimerkki (Kuvio 14) on puolestaan Snortin Full-hälytystoiminnolla tehty loki-merkintä. Jos kuvaa verrataan Snortin Fast-hälytyksen tekemään merkintään (Kuvio 13), on tässä paljon enemmän dataa tallennettuna. Full-hälytyksessä Snort tallettaa lokiin täydelliset otsikkotiedot paketeista. On hyvä, että saadaan mahdollisimman paljon tietoa tunkeilijasta, mutta tietojen tallentaminen vie myös prosessori tehoa. Full-hälytystoiminto purkaa paketin otsikkotiedot ASCII-merkinnäksi ja tallettaa ne loki-tiedostoon. Jos verkossa on paljon liikennettä, Snortilla menee turhaa prosessorin resursseja tiedon tallettamiseen. Siksi suositaan Fast-hälytystoimintoja, joka tallettaa vähemmän tietoa lokiin, mutta on toisaalta huomattavasti nopeampi, ja säästää Snortin resursseja pakettien analysointiin. (Hayes 2004.)

```

[**] [1:1668:5] WEB-CGI /cgi-bin/ access [**]
[Classification: Web Application Attack] [Priority: 1]
01/16-23:06:11.675382 0:10:67:0:B2:50 -> 0:A0:CC:D2:10:31 type:0x800 len:0xDD
192.168.1.68:44561 -> 172.16.34.18:80 TCP TTL:47 TOS:0x0 ID:53932 IpLen:20
      DgmLen:207 DF
***AP*** Seq: 0xD5349716 Ack: 0x2B34D8BB Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 322913730 135653681

[**] [1:1201:7] ATTACK-RESPONSES 403 Forbidden [**]
[Classification: Attempted Information Leak] [Priority: 2]
01/16-23:06:11.675864 0:A0:CC:D2:10:31 -> 0:10:67:0:B2:50 type:0x800 len:0x268
172.16.34.18:80 -> 192.168.1.68:44561 TCP TTL:64 TOS:0x0 ID:12687 IpLen:20
      DgmLen:602 DF
***AP*** Seq: 0x2B34D8BB Ack: 0xD53497B1 Win: 0x16A0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 135653685 322913730

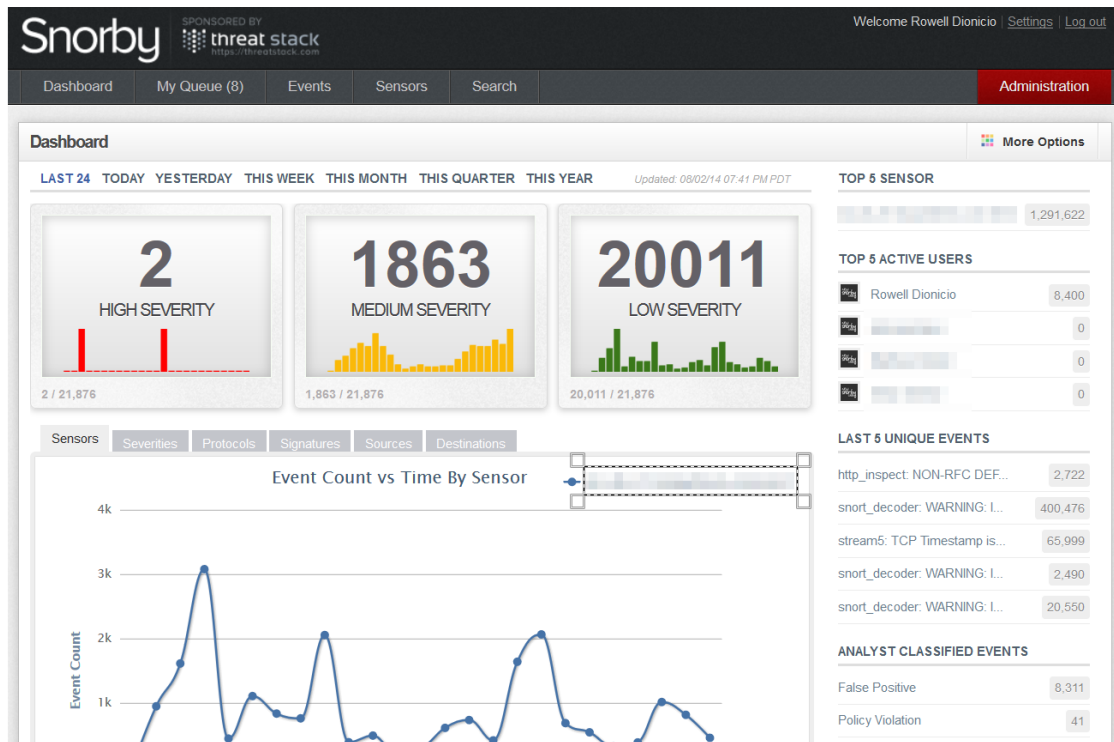
[**] [1:1852:3] WEB-MISC robots.txt access [**]
[Classification: access to a potentially vulnerable Web application] [Priority:
  2]
01/16-23:06:13.035036 0:10:67:0:B2:50 -> 0:A0:CC:D2:10:31 type:0x800 len:0xDF
192.168.1.68:44572 -> 172.16.34.18:80 TCP TTL:47 TOS:0x0 ID:27543 IpLen:20
      DgmLen:209 DF
***AP*** Seq: 0xD5D72DD3 Ack: 0x2BD8E79A Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 322913866 135653817
[Xref => http://cgi.nessus.org/plugins/dump.php?id=10302]

```

Kuvio 14. Toinen esimerkki Snortin loki-merkinnästä (Hayes 2004)

Snort pystyy kirjoittamaan tietoja talteen tcpdump binääri-merkinällä, jolloin kirjoitus on todella nopeaa, ja Snort saa varmimmin jokaisen paketin luettua. Unified2-lisäosa ottaa Snortilta raakadatan itsellensä, ja tallettaa tiedot binäärimuotoisena. Binäärimuotoinen talletus ei kuitenkaan hivele ihmissilmää, sillä se koostuu kokonaisuudessaan vain ykkösistä ja nolista. Barnyard2 on erillinen ohjelma, joka kykenee muuttamaan binäärimuotoisen datan ihmiselle luettavaan muotoon, ja tallettamaan tiedot käyttäjän valitsemaan sijaintiin. Sijainti voi olla loki-tiedosto, syslog, WEB-sivusto, tai MySQL-tietokanta. (Hayes 2004.)

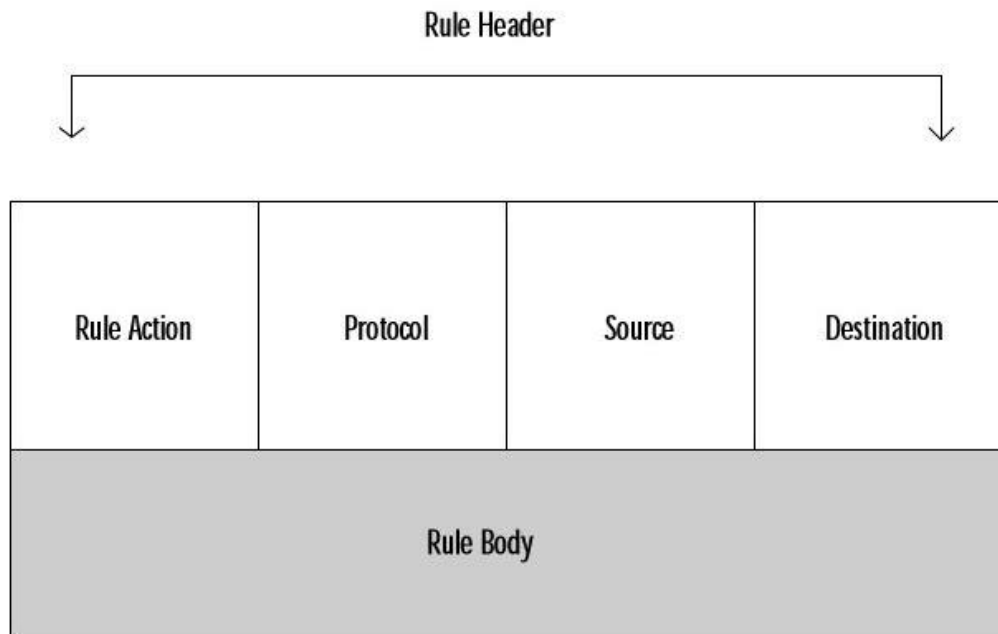
Snortin kehittäjät suosittelevat käyttämään Snorby- tai BASE-tietokantoja, kun dataa halutaan tallettaa tietokantaan. Molemmilla tietokannoilla on graafinen WEB-käyttöliittymä, joka on selkeä ja helppokäyttöinen. Tietokannan etu on merkittävä, kun halutaan tutkia hälytyksiä tarkemmin ja saada esille tilastotietoja. Kuviossa 15 on kuvankaappaus Snorbyn WEB-käyttöliittymän etusivulta, jossa on esitetty tiivistetysti viimeaikaiset hälytykset.



Kuvio 15. Snorby Dashboard (Dionicio 2014)

6.3 Snort-säännön rakenne

Snort-sääntö koostuu otsikkotiedosta ja sisällöstä. Kuvioista 16 nähdään, kuinka otsikkotiedot (Rule header) voidaan jakaa neljään eri osaan. Heti otsikkotiedon alussa määritetään toiminto (Rule Action), joka suoritetaan, jos vastaavuus sääntöön löytyy. Toiminto voi olla hälytys (Alert), talletus lokiin (Log) tai ohittaa paketin (Pass). Sääntö voi myös aktivoida (Activate) toisen säännön, joka myös käy paketin läpi. Ohita paketti eli Pass-sääntöä voidaan käyttää silloin, jos Snort tekee False Positive-hälytyksiä eli hälytyksiä sallitusta liikenteestä. (Alder 2004.)



Kuvio 16. Snort säännön rakenne (Alder 2004)

Snort säännön otsikkotietoon voidaan määrittää

- käytetty protokolla (TCP, UDP, IP, ICMP)
- lähettäjän ja/tai vastaanottajan portti
- lähettäjän ja/tai vastaanottajan IP-osoite/verkko
- liikenteen suunta.

Otsikkotietojen jälkeen säännössä on itse säännön sisältö. Sisällön perusteella voidaan tarkentaa sääntöä, muun muassa määrittämällä yksityiskohtaisemmin asioita, joita paketeista etsitään. (Alder 2004.)

Kuviossa 17 on esimerkki Snort-säännöstä. Säännön alussa määritetään toiminto, joka tässä on Alert. Toisin sanoen halutaan hälytys, kun vastaavuus sääntöön löytyy. Säännön otsikkotiedoissa määritetään protokolla, joka on ICMP sekä paketin lähettäjän IP-osoite, 192.168.1.4. IP-osoitteen jälkeen määritetään portti, josta viesti on lähetetty. ICMP-viestissä lähettäjän ja vastaanottajan porttia ei ole määritetty, siksi portin numeron tilalla on sana "any". Se merkitsee sitä, ettei portin arvolla ole merki-

tystä. Seuraavaksi säännössä on määritetty nuoli, joka määrittää paketin kulku suunnan. Tässä siis IP-osoitteella 192.168.1.1 oleva osapuoli on viestin vastaanottaja. (Ur Rehman 2003.)

```
alert icmp 192.168.1.4 any -> 192.168.1.1 any (msg:
"HEARTBEAT";)
```

Kuvio 17. Esimerkki Snort säännöstä (Rehman 2003)

Sulkeissa oleva tieto on säännön sisältöä. Tämän säännön sisällössä ei määritetä tarkempia ehtoja, jotka tulee täyttyä hälytyksen laukaisemiseksi. Sen sijaan määritetään viesti, joka kirjautuu hälytyksen yhteyteen. Tämän kaltaista sääntöä voidaan käyttää testi tarkoituksessa, kun halutaan varmistaa Snortin toimivuus. Kun ajastetaan IP-osoitteella 192.168.1.4 oleva laite lähettämään säännöllisesti ICMP-paketti 192.168.1.1-osoitteella varustetulle laitteelle, Snortin tehdessä paketista hälytyksen, voi verkon hallinnoija olla varma että Snort toimii. (Rehman 2003.)

6.3.1 Sisällön vaihtoehdot

Snort säännön sisältöä avulla voidaan tarkentaa sääntöä vastaamaan paremmin etsittäväan pakettiin. Sisältöön voidaan määrittää esimerkiksi sanoja, joita etsitään paketin kuormaosasta (eng. payload). Sanoja, joita usein etsitään, ovat esimerkiksi "root", "/etc/passwd" ja "GET". Näitä sanoja verkkotunkeilijat käyttävät murtautukseen järjestelmiin, tai etsiessään järjestelmistä tietoja. Kuviossa 18 oleva Snort-sääntö on esimerkki, jossa halutaan luoda hälytys, kun verkossa liikkuu TCP-paketti, jonka kuormaosasta löytyy sanat "/etc/passwd". Kun Snort havaitsee tällaisen paketin verkossa, se antaa hälytyksen, ja kirjaa hälytyksen yhteyteen viestin "Searching for ASCII Gargabe!". (Rehman 2003.)

```
alert tcp any any -> any any (content: "/etc/passwd";  
msg:"Searching for ASCII Garbage!");
```

Kuvio 18. Toinen esimerkki Snort säännöstä

TCP-yhteyksissä käytetään lippuja, jota yhteyden osapuolet käyttävät kommunikoinnin tukena. Lippujen käytetään esimerkiksi yhteyttä avattaessa ja suljettaessa. Niitä käytetään myös ilmoittamaan tulevista paketeista, pakettien kiireellisyydestä sekä kuittaamaan jo saapuneita paketteja. Verkkotunkeilijat käyttävät näitä lippuja hyväkseen pyrkimällä katkaisemalla yhteyden (Man in The Middle-hyökkäys), tai saadaakseen tietoja verkosta. Tällaisten hyökkäysten varalta Snortin pystyy tarkkailemaan TCP-paketteihin asetettuja lippuja, ja antamaan hälytyksen, jos havaitsee epätavallisia lippumerkintöjä. TCP-pakettien lisäksi Snort pystyy tarkastelemaan lähemmin myös IP- ja ICMP-pakettien sisältöjä. (Hayes 2004.)

Snortin sääntöjä voidaan luokitella eri luokkiin, ja jokaiselle luokalle voidaan antaa oma prioriteetti. Sääntöjen luokittelu eri aihealueisiin helpottaa hyökkäysten tutkimista jälkikäteen, ja verkon hallinnoija saa paremman kuvan siitä, mitä verkossa tapahtuu. Luokkien priorisoinnilla voidaan määrittää, missä järjestyksessä sääntöjä käydään läpi, ja minkälaisia toimenpiteitä tehdään kun vastaavuus löytyy. Lisäksi priorisoinnilla määritetään hälytyksen näkyvyys. Pienellä prioriteetilla olevat hälytykset kirjoitetaan vaan lokiin, mutta verkkoa todennäköisesti uhkaava hälytys tulostetaan konsolille. Tällöin verkon hallinnoija tietää heti, että verkkoa uhkaava hyökkäys on käynnissä. (Rehman 2003.)

7 Työn toteutus

Työ toteutetaan täysin virtuaalisessa ympäristössä. Virtualisoinnin alustana toimii Windows-käyttöjärjestelmässä ajettava VMWare Workstation Player 12, jonka avulla virtuaalisoidaan Linux-palvelimia. Työtä varten on luotu kaksi virtuaalikonetta, joihin

on esi-asennettuna puhtaat Linux Ubuntu Server 14.04-käyttöjärjestelmät. Työn toteutuksessa asennetaan Ryu-kontrolleri, Mininet (verkon virtualisointi-ohjelma) sekä Snort-tunkeilijan estojärjestelmä. Mininet ja Snort asennetaan samalle virtuaalikoneelle. Ryu on asennettuna omalle koneelleen. Integraatio toteutetaan valmiiksi olemassa olevilla applikaatioilla, jotka ovat ladattavissa GitHub-palvelimelta.

7.1 Asennus

7.1.1 Ryu

Ryu asennetaan yksin omalle virtuaalikoneelle. Ennen Ryun asentamista, asennetaan kaikki Ryun vaatimat paketit. Koska Ryu on kirjoitettu Python-ohjelmointikielellä, tarvitsee se toimiakseen useita Python-paketteja ja -kirjastoja. Näiden avulla mahdollistetaan Python-ohjelmien toiminta ja yhteensopivuus muiden komponenttien kanssa. Taulukossa 2 on listaus tarvittavista paketeista ja niiden käyttötarkoituksesta.

Taulukko 2. Ryun vaatimat lisäosat

Paketin nimi	Käyttötarkoitus
lxml	Python XML – prosessointi työkalu
paramiko	Mahdollistaa SSH-salauksen
eventlet	Lisätyökalu Python – ohjelmien ajamiseen
msgpack	Tietoliikennepakettien (MessagePacket) datan kirjoittamiseen ja lukemiseen käytetty työkalu
netaddr	Tietoverkko-osoitukseen käytetty työkalu
oslo.config	Komentorivin argumenttien käsittelytyökalu
routes	URL-reititystietojen tunnistus ja luonti työkalu
six	Mahdollistaa eri Python-versioiden yhteensopivuuden
webob	HTTP-toiminnollisuus

Ryun tarvitsemat lisäosat asennetaan Python-Pip-asennustyökalun avulla. Työkalun saa asennettua APT-pakettienhallintatyökalun avulla. Koska kyseessä on Python-paketti, tarvitaan asennuksen tueksi myös Python-hallintatyökalu. Seuraavalla komennolla voidaan asentaa sekä Python-hallintatyökalu (python-setuptools) että Pip-paketinhallintatyökalu:

```
sudo apt-get install python-setuptools python-pip
```

mukaiset paketit voidaan asentaa komennolla:

```
sudo pip install lxml paramiko eventlet msgpack-python  
netaddr oslo.config routes six webob
```

Kun Python-ominaisuuksien mahdollistavat paketit on asennettu, voidaan itse Ryu asentaa. Ryu kannattaa asentaa GitHub-palvelimelta viimeisimpien muutosten saamiseksi. Git-työkalun voi asentaa APT-työkalun avulla komennolla:

```
sudo apt-get install git-all
```

Kun Git on asentunut, voidaan ladata Ryun lähdekoodi palvelimelta. Lähdekoodi latautuu oletuksena siihen tiedostosijaintiin, jossa komento suoritetaan.

```
sudo git clone git://github.com/osrg/ryu.git
```

Siirytään ryu-kansioon ja suoritetaan asennus:

```
cd ryu  
sudo python ./setup.py install
```

Ryun toimivuuden voi varmistaa siirtymällä ryu/app-kansioon ja suorittamalla simple_switch_12-aplikaation komennolla:

```
cd ryu/app  
sudo ryu-manager simple_switch_12.py
```

Jos ohjelman alustus onnistuu ilman virheitä (katso Kuvio 19), on asennus onnistunut. Jos Ryu ilmoittaa virheistä, tulee virheen mukaiset paketit asentaa tai päivittää.

```
root@ubuntu:/home/maria/ryu/ryu/app# ryu-manager ./simple_switch_12.py
loading app ./simple_switch_12.py
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ./simple_switch_12.py of SimpleSwitch12
_
```

Kuvio 19. Ryun applikaation onnistunut käynnistäminen

7.1.2 Mininet

Mininet asennetaan eri virtuaalikoneelle kuin Ryu. Mininet on verkon virtualisointityökalu. Mininet on ladattavissa myös GitHub-palvelimelta. Alla olevalla komennolla voidaan ladata Mininetin lähdekoodi.

```
git clone git://github.com/mininet/mininet.git
```

Mininetistä on saatavilla useampi versio. Ennen Mininetin asentamista, tulee valita haluttu versio komennolla: `git checkout <versio numero>`. Asennus tapahtuu seuraavanlaisesti:

```
cd mininet
git tag
git checkout 2.2.1
util/install.sh -fnv
```

Mininetin mukana asentuu myös OpenFlow-protokolla sekä OpenvSwitch-ohjelmisto. Mininetin onnistuneen asennuksen voi varmistaa ajamalla komennon:

```
mn -test pingall
```

Kun testin tulos on Kuvion 20 mukainen, on asennus onnistunut

```

root@ubuntu:/home/maria# mn --test pingall
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 switches
s1 ..
*** Stopping 2 hosts
h1 h2
*** Stopping 1 controllers
c0
*** Done
completed in 0.767 seconds

```

Kuvio 20. Mininet-verkon testaaminen

7.1.3 Snort

Snort asennetaan samalle virtuaalikoneelle kuin Mininet. Ennen Snortin asentamista, asennetaan Snortin vaatimat paketit. Snort tarvitsee muun muassa libcap-lisäosan, jonka avulla se kykenee kaappaamaan tietoliikennepaketteja medialta. Alla olevalla komennolla voidaan asentaa kaikki Snortin vaativat lisäosat.

```

sudo apt-get install flex bison build-essential
checkinstall libpcap-dev libnet1-dev libpcrc3-dev
libmysqlclient15-dev libnetfilter-queue-dev iptables-dev

```

Lisäksi Snort tarvitsee DAQ:n (Data Acquisition-kirjasto), joka toimii lisätyökaluna Libcap-kirjaston tukena. Se voidaan asentaa pakatusta tiedostosta, joka on saatavilla Snort.org-verkkosivulta. (Snort user manual 2015.)

```

wget https://www.snort.org/downloads/snort/daq-2.0.6.tar.gz
tar xvfz daq-2.0.6.tar.gz
cd daq-2.0.6
./configure; make; sudo make install

```

Samalla tavalla voidaan ladata ja asentaa Snort

```
wget https://www.snort.org/downloads/snort/snort-2.9.8.0.tar.gz
tar xvfz snort-2.9.8.0.tar.gz
cd snort-2.9.8.0
./configure --enable-sourcefire; make; sudo make install
```

Snort on nyt asennettu, mutta tarvitaan vielä konfiguraatio muutoksia, jotta Snort toimisi odotetusti. Seuraavilla komennoilla Snort päivittää kirjastonsa, sekä luo symbolisen linkin Snort binääri-tiedostoon.

```
sudo ldconfig
sudo ln -s /usr/local/bin/snort /usr/sbin/snort
```

Snortin toiminnan voi testata komennolla `snort -v`, jonka avulla Snort itse varmistaa toiminnollisuutensa. Kuvio 21 esittää tuloksen onnistuneesta asennuksesta.

```

,,_
o" )~
''''
--> Snort! <*-
Version 2.9.8.0 GRE (Build 229)
By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014-2015 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using libpcap version 1.5.3
Using PCRE version: 8.31 2012-07-06
Using ZLIB version: 1.2.8

```

Kuvio 21. Snortin asennuksen varmistaminen

Snort tarvitsee oikeudet ajaa ohjelmia, jonka vuoksi Snortille luodaan oma käyttäjä ja käyttäjäryhmä. Lisäksi Snort tarvitsee hakemistorakenteen, jonne se tallettaa loki-tiedostoja, ja josta se etsii sääntöjä. Alla olevan ohjeen mukaan voidaan luoda Snortille tarvittava hakemistorakenne sekä oikeudet käyttää sitä. (Dietrich 2015.)

```
# Luodaan Snort käyttäjä ja käyttäjäryhmä:
sudo groupadd snort
sudo useradd snort -r -s /sbin/nologin -c SNORT_IDS
-g snort
```

```

# Luodaan hakemistorakenne:
sudo mkdir /etc/snort
sudo mkdir /etc/snort/rules
sudo mkdir /etc/snort/rules/iplists
sudo mkdir /etc/snort/preproc_rules
sudo mkdir /usr/local/lib/snort_dynamicrules
sudo mkdir /etc/snort/so_rules

# Luodaan Snortin tarvitsemat tiedostoja:
sudo touch /etc/snort/rules/iplists/black_list.rules
sudo touch /etc/snort/rules/iplists/white_list.rules
sudo touch /etc/snort/rules/local.rules
sudo touch /etc/snort/sid-msg.map

# Luodaan loki-hakemisto
sudo mkdir /var/log/snort
sudo mkdir /var/log/snort/archived_logs

# Lisätään Snortille oikeuksia
sudo chmod -R 5775 /etc/snort
sudo chmod -R 5775 /var/log/snort
sudo chmod -R 5775 /var/log/snort/archived_logs
sudo chmod -R 5775 /etc/snort/so_rules
sudo chmod -R 5775 /usr/local/lib/snort_dynamicrules

# Muutetaan kansioiden omistajuus
sudo chown -R snort:snort /etc/snort
sudo chown -R snort:snort /var/log/snort
sudo chown -R snort:snort /usr/local/lib/snort_dynamicrules

# Kopioidaan tarvittavat konfiguraatio-tiedostot luotuun
hakemistorakenteeseen. (~snort_src viittaa tiedostosijain
tiin, jonne Snort alun perin ladattiin)

cd ~/snort_src/snort-2.9.8.0/etc/
sudo cp *.conf* /etc/snort
sudo cp *.map /etc/snort
sudo cp *.dtd /etc/snort
cd ~/snort_src/snort-2.9.8.0/src/dynamic-preprocessors
/build/usr/local/lib/snort_dynamicpreprocessor/

sudo cp * /usr/local/lib/snort_dynamicpreprocessor/

```

Snortin tärkeimmässä konfiguraatio-tiedostossa (snort.conf) on asetuksia, joita muokkaamalla saadaan Snort toimimaan halutulla tavalla. Oletuksena kaikki Snortin ominaisuudet on poissa käytöstä. Snortiin on laadittu valmiiksi suuri määrä sääntöjä, joilla voidaan turvata verkon toimintaa. Säännöt voidaan ottaa käyttöön poistamalla kommenttimerkintä (#) sääntötiedoston edestä. Jokaisen kommenttimerkin voi poistaa yksitellen, tai ne voi poistaa kaikki kerralla komennolla:


```
sudo sed -i "s/include \$RULE_PATH/#include \$RULE_PATH/"
/etc/snort/snort.conf
```

Muokataan vielä snort.conf-tiedostoa. Avataan tiedosto tekstinkäsittelyohjelmalla, esimerkiksi:

```
nano /etc/snort/snort.conf
```

Tiedosto sisältää muuttujia, joiden avulla eritetään ”oma verkko” ulkopuolisesta verkosta. Lisätään muuttujaan HOME_NET oman verkon osoite. Muuttuja sijaitsee rivillä 45. Lisäksi voidaan lisätä muuttujaan EXTERNAL_NET arvo: !\$HOME_NET. Huutomerkillä tarkoitetaan kaikkia muita verkkoja paitsi muuttujan HOME_NET mukaista verkkoa. Kuvio 22 näyttää esimerkkiä.

```
#####
# Step #1: Set the network variables. For more information, see README.variables
#####

# Setup the network addresses you are protecting
ipvar HOME_NET 10.0.0.0/24

# Set up the external network addresses. Leave as "any" in most situations
ipvar EXTERNAL_NET !$HOME_NET
```

Kuvio 22. Snortin muuttujien lisääminen

Lisätään vielä hakemistosijainnit säännöille. Muokataan rivejä 104 alkaen Kuvion 23 mukaisesti

```

# Path to your rules files (this can be a relative path)
# Note for Windows users: You are advised to make this an absolute path,
# such as: c:\snort\rules
var RULE_PATH /etc/snort/rules
var SO_RULE_PATH /etc/snort/so_rules
var PREPROC_RULE_PATH /etc/snort/preproc_rules

# If you are using reputation preprocessor set these
# Currently there is a bug with relative paths, they are relative to where snort is
# not relative to snort.conf like the above variables
# This is completely inconsistent with how other vars work, BUG 89986
# Set the absolute path appropriately
var WHITE_LIST_PATH /etc/snort/rules/iplists
var BLACK_LIST_PATH /etc/snort/rules/iplists

```

Kuvio 23. Sääntöjen tiedostohakemiston lisääminen

Ja lopuksi tiedostosijainti (riviltä 545 alkaen), jonne voidaan lisätä omia sääntöjä (katso Kuvio 24). Lisätään sääntötiedosto ryu.rules, jonne lisätään testiympäristön sääntöjä.

```

#####
# Step #7: Customize your rule set
# For more information, see Snort Manual, Writing Snort Rules
#
# NOTE: All categories are enabled in this conf file
#####

# site specific rules
include $RULE_PATH/local.rules
include $RULE_PATH/ryu.rules

```

Kuvio 24. Include -tiedostosijaintien lisääminen

Nyt kun asetukset ovat valmiita, varmistetaan konfiguraatio-tiedostojen virheettömyys komennolla:

```
sudo snort -T -i eth0 -c /etc/snort/snort.conf
```

Lopputuloksen tulee olla Kuvion 25 mukainen.

```
Snort successfully validated the configuration!  
Snort exiting  
root@ubuntu:/etc/snort# _
```

Kuvio 25. Snort-konfiguraatiotiedoston varmistaminen

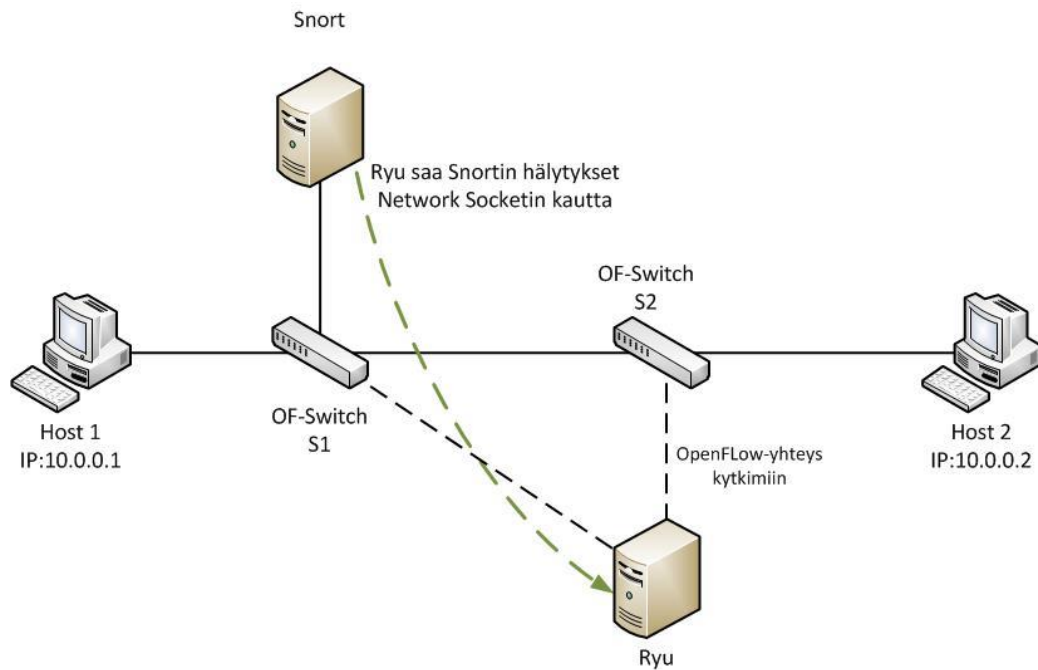
Snortin lisäksi tarvitaan vielä Pigrelay-applikaatio, jonka tehtävä on välittää Snortilta saamansa hälytykset Ryu-kontrollerille. Pigrelay-applikaation lähdekoodi on ladattavissa GitHub-palvelimelta. Pigrelay kannattaa sijoittaa Snort-hakemistorakenteeseen.

```
# Luodaan Pigrelay-applikaatiolle oma kansio  
cd /etc/snort  
mkdir pigrelay
```

```
# Siirrytään kansioon ja ladataan lähdekoodi  
cd pigrelay  
git clone git://github.com/John-Lin/pigrelay
```

7.2 Ensimmäinen testi

Testiverkossa on kaksi OpenFlow-kytkintä (OF-Switch), jotka välittävät tietoliikennepaketteja Host 1:n ja Host 2:n välillä. Verkkokontrollerina toimii Ryu. Kuvio 26 esittää loogista verkkotopologiaa. Snort kuuntelee kytkimen S1 porttia s1-eth2, ja kaappaa medialta jokaisen näkemänsä paketin. Snort analysoi niitä etsien vastaavuuksia etukäteen luotuihin sääntöihin. Kun vastaavuus sääntöön löytyy, Snort välittää siitä tiedon Ryu-kontrollerille.



Kuvio 26. Ensimmäisen testin verkkotopologia

Luodaan Mininet-työkalulla virtuaalinen lähiverkko, jossa kontrolleri on Mininet-verkon ulkopuolella. Vaikka Mininet-verkon päätelaitteet ja kontrolleri ovat eri verkoissa, ei reititykseen tarvitse ottaa kantaa. Reititys tapahtuu virtuaaliympäristön ulkopuolella. Voidaan kuitenkin loogisella tasolla ajatella tietoverkon olevan Kuvion 26 mukainen. Luodaan Mininet verkko alla olevalla komennolla. Komennon argumentit on selitetty Taulukossa 3.

```
sudo mn --topo linear,2 --switch ovsk,protocols=OpenFlow13
--controller remote,ip=192.168.1.80,port=6633
```

Taulukko 3. Mininet-komennon argumentit

Argumentti	Selitys
--topo	Määritetään topologia
linear,2	Lineaarinen topologia, kaksi kytkintä ja kaksi päätelaitetta
--switch	Määritetään kytkin
ovsk	Kytkimet ovat OVS-kytkimiä
protocols	Määritetään käytetty protokolla (tässä OpenFlow v.1.3)
--controller	Määritetään kontrolleri
remote	Kontrolleri on Mininet verkon ulkopuolella
ip	Kontrollerin IP-osoite
port	Kontrollerin portti

Konsolilla lopputuloksen tulee olla Kuvion 27 mukainen.

```

root@ubuntu:/etc# sudo mn --topo linear,2 --switch ovsk,protocols=OpenFlow13 --controller remote,ip=
192.168.1.80,port=6633
*** Creating network
*** Adding controller
*** Adding controller
Unable to contact the remote controller at 192.168.1.80:6633
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s2) (s2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
mininet>

```

Kuvio 27. Mininet testiverkon luominen

Avataan uusi terminaali-ikkuna samassa virtuaalikoneessa näppäinyhdistelmällä Alt + F2. Lisätään Snortiin sääntö, jonka mukaan se tekee hälytyksen, kun se havaitsee

ICMP-paketin. Avataan tiedosto `/etc/snort/rules/ryu.rules` ja lisätään sinne seuraavanlainen rivi.

```
alert icmp any -> any any (msg:"Pinging... ";
sid:1000004;)
```

Tarkoituksena on kuunnella OpenFlow-kytkimien välistä liikennettä. Asetetaan kytkimen S1 rajapinta, promiscuous-tilaan komennolla:

```
sudo ifconfig s1-eth2 promisc
```

Siirrytään Ryu-virtuaalikoneelle, jolla ajetaan applikaatio `simple_switch_snort.py`. Ohjelma sijaitsee tiedostosijainnissa `ryu/app`. Kyseisestä applikaatiota tulee muokata niin, että ohjelma kuuntelee sille Network Socketin kautta tulevia tapahtumia. Muokataan riviä 43 `unixsock : False`. Kuvio 28 näyttää esimerkkiä oikeasta konfiguraatiosta

```
socket_config = {'unixsock': False}
```

Kuvio 28. Unixsock-toiminnon asettaminen

Suoritetaan ohjelma komennolla:

```
sudo ryu-manager simple_switch_snort.py
```

Komennon suorittaminen konsolilla tulostuu Kuvion 29 mukaiset tiedot.

```

ryu@ubuntu:~/ryu/ryu/app$ ryu-manager simple_switch_snort.py
loading app simple_switch_snort.py
loading app ryu.controller.ofp_handler
instantiating app None of SnortLib
creating context snortlib
instantiating app simple_switch_snort.py of SimpleSwitchSnort
[snort][INFO] {'port': 51234, 'unixsock': False}
instantiating app ryu.controller.ofp_handler of OFPHandler
[snort][INFO] Network socket server start listening...
-

```

Kuvio 29. Simple_switch_snort.py -applikaation suorittaminen

Käynnistetään Snort samalla virtuaalikoneella kuin Mininet, terminaalissa 2. Komennon argumentit on selitetty Taulukossa 4.

```
snort -i s1-eth2 -A unsock -l /tmp -c /etc/snort/snort.conf
```

Taulukko 4. Snort-komennon argumentit

Argumentti	Selitys
-i s1-eth2	Rajapinta, jota kuunnellaan
-A unsock	Hälytys metodi (Unsock = välitä hälytys toiselle UNIX-järjestelmälle)
-l /tmp	Loki-tiedoston sijainti
-c /etc/snort/snort.conf	Konfiguraatio-tiedosto, josta asetukset ladataan

Suoritetaan samalla virtuaalikoneella myös Pigrelay-applikaatio. Applikaation lähdekoodiin tulee lisätä Ryu-kontrollerin IP-osoite, jotta se osaa välittää tiedon hälytyksestä oikealle laitteelle. Voit avata uuden terminaali-ikkunan näppäinyhdistelmällä Alt + F3. Avaa /etc/snort/pigrelay/pigrelay.py tekstinkäsittelyohjelmalla, ja lisää riville 20 kontrollerin IP-osoite. Kuvio 30 näyttää esimerkkiä.

```
sudo nano /etc/snort/pigrelay/pigrelay.py
```

```
# Must to set your controller IP here
CONTROLLER_IP = '192.168.1.80'

# Controller port is 51234 by default.
# If you want to change the port number
# you need to set the same port number in the controller application.
CONTROLLER_PORT = 51234
```

Kuvio 30. Pigrelay-applikaation muokkaaminen.

Suorita pigrely komennolla:

```
sudo python pigrelay.py
```

Ryu-koneelta voidaan varmistaa, että yhteys Pigrelay-applikaation ja Ryu-kontrollerin välillä on muodostettu. Onnistuneen yhteyden muodostuksen jälkeen Ryu-konsolille ilmestyy Kuvion 31 mukainen ilmoitus.

```
[snort][INFO] Network socket server start listening...
[snort][INFO] Connected with 192.168.1.79
```

Kuvio 31. Pigrelayn ja kontrollerin yhteyden varmentaminen

Lähetetään ICMP-paketti Host 1:lta Host 2:lle. Avataan terminaali, jossa Mininet-applikaatio on, ja suoritetaan komento:

```
pingall
```

Kun katsotaan Ryu-konsolia, saa se pigrelay-applikaation välityksellä tiedon hälytyksestä. Ryu purkaa ja tulostaa tiedot hälytyksen synnyttäneestä paketista. Tuloste on Kuvion 32 mukainen.


```

alertmsg: Pinging...
icmp(code=0,csum=52170,data=echo(data=array('B', [196, 51, 216, 86, 144, 34, 14, 0, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55]),id=1668,seq=1),type=8)
ipv4(csum=63737,dst='10.0.0.1',flags=2,header_length=5,identification=11693,offset=0,option=None,proto=1,src='10.0.0.2',tos=0,total_length=84,ttl=64,version=4)
ethernet(dst='42:ab:39:f6:18:b7',ethertype=2048,src='fa:05:19:03:1e:7d')
alertmsg: Pinging...
icmp(code=0,csum=54218,data=echo(data=array('B', [196, 51, 216, 86, 144, 34, 14, 0, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55]),id=1668,seq=1),type=0)
ipv4(csum=5463,dst='10.0.0.2',flags=0,header_length=5,identification=20816,offset=0,option=None,proto=1,src='10.0.0.1',tos=0,total_length=84,ttl=64,version=4)
ethernet(dst='fa:05:19:03:1e:7d',ethertype=2048,src='42:ab:39:f6:18:b7')
-

```

Kuvio 32. Hälytys Ryu-kontrollerilla

Kuvion 32 perusteella nähdään, että Ryu saa tiedon verkosta kulkevasta ICMP-paketista. Jos Pingall-komento ajetaan uudelleen Host 1:lta Host 2:lle, saa Ryu siitäkin samanlaisen hälytyksen. Ryu siis purkaa ja tulostaa ICMP-paketin tiedot, muttei reagoi muuten saamaansa hälytykseen. Voidaan siis todeta, että integraatio Ryu-kontrollerin ja Snort tunkeilijan estojärjestelmän välillä on olemassa, mutta se on hyvin puutteellinen.

7.3 Integraation tarkastelu

7.3.1 Applikaation lähdekoodin tarkastelu

Tutkitaan hieman `simple_switch_snort.py`-applikaation lähdekoodia. Lähdekoodi on esitetty kokonaisuudessa Liitteessä 1.

Lähdekoodin alussa määritellään kirjastoja ja paketteja, joita tarvitaan ohjelman suorittamisessa. Jokainen ohjelma periytyy `RyuApp`-luokasta, joka tarjoaa applikaatioille perustoiminnallisuuden. `RyuApp`-luokkaan on määritetty tietyn OpenFlow-protokollan versio, sekä Snort-kirjasto, jota kaikki applikaation alaluokat voivat käyttää. Applikaation ensimmäisessä alaluokassa määritellään Network Socketin asetuksia. Koska Snort voidaan ajaa samalla laitteella kuin Ryu, on konfiguraatiossa määritetty myös Snort-portti (`snort_port = 3`). Tätä ominaisuutta ei kuitenkaan käytetty tässä opinnäytetyössä. `'Socket_config = False'` avulla määritetään, että hälytykset tulevat toiselta UNIX-järjestelmältä.

Applikaatio sisältää EventAlert-tapahtumankäsittelijän, jota kutsutaan aina, kun applikaatio saa hälytyksen Snortilta Network Socketin kautta. Tapahtumankäsittelijä purkaa saamastaan tapahtumasta hälytysviestin, tulostaa sen konsolille ja välittää viestin sisältämän paketin (msg.pkt) edelleen packet_print-luokalle. Packet_print-luokan tehtävä on purkaa paketti loppuun ja tulostaa sen sisältö konsolille.

Loppu osassa applikaatiota määritellään OpenFlow-kytkimen perustoimintaa. Switch_features_handler-alaluokassa määritetään, kuinka yhteys kontrollerin ja kytkimen välille muodostetaan. Kontrolleri sopii kytkimen kanssa yhteyden aikaisista ominaisuuksista, kuten esimerkiksi käytetystä OpenFlow-versiosta. Yhteyden muodostamisen jälkeen, kontrolleri luo ensimmäisen vuosäännön kytkimelle, joka on 'actions = ofproto.OFP_CONTROLLER'. Tällä säännöllä kontrolleri ohjeistaa kytkimiä välittämään paketit kontrollerille, jotta voi sitten antaa tarkemmat ohjeet pakettien välittämiseksi jatkossa.

EventPacketIn"-tapahtumankäsittelijässä määritetään, kuinka kontrolleri käsittelee kytkimeltä saapuvia viestejä. Tapahtumankäsittelijää kutsutaan, kun kontrolleri vastaan ottaa PacketIn-viestin kytkimeltä. Yleensä kytkin ottaa yhteyttä kontrolleriin vain, kun se vastaanottaa paketin, jota se ei osaa välittää eteenpäin. Toisin sanoen, tässä luokassa määritetään perinteisen verkkokytkimen toimintaa. Kontrolleri tallentaa tiedot MAC-osoitteista, porteista ja datapath-ID:stä mac_to_port-tauluun. Tämä taulu toimii samalla lailla, kuin CAM-taulu perinteisessä kytkimessä. Kontrolleri tutkii mac-to-port-taulua etsien sieltä vastaavuutta paketin vastaanottajan MAC-osoitteeseen. Jos vastaavuus löytyy, kontrolleri välittää tiedot add_flow-luokalle, joka luo uuden vuosäännön, ja välittää PacketOut-viestin kytkimelle. Jos vastaavuutta ei löydy, antaa kontrolleri ohjeen välittää paketti ulos kaikista porteista ('out_port = ofproto.OFPP_FLOOD').

7.3.2 Integraation kehittäminen

Kuten aikaisemmin jo todettiin, on simple_switch_snort-applikaatio puutteellinen. Applikaatio ainoastaan tulostaa hälytyksen laukaisseeseen paketin sisällön, ja mahdollistaa kytkimien perustoiminnan. Se ei tee mitään estääkseen hälytyksen toistumista.

Applikaation lähdekoodin tarkemman tutkimisen myötä voidaan todeta, että integraatiota on lähdetty kehittämään, mutta sen kehitys on syystä tai toisesta jäänyt kesken.

Lähdetään pohtimaan, kuinka integraatiota voitaisiin edelleen kehittää. Ryu saa hälytyksen Snortilta ja sen mukana tiedot hälytyksen aiheuttaneesta paketista. Paketti sisältää tietoja, joita voidaan käyttää pohjana uuden vuosäännön kehittämiseen. Kuitenkin, jotta uusi vuosääntö vastaisi mahdollisimman tarkasti Snort-sääntöä, on tehokkaampaa käyttää Snort-sääntöä uuden vuosäännön pohjana. Snort ei kuitenkaan lähetä Ryulle mitään tietoa säännöstä, joten integraation toimintaa tulee hieman muokata, jotta Ryu pääsee käsiksi Snortin sääntöihin.

Snort tukee erilaisia hälytysten talletusmuotoja, joita voidaan käyttää tässä hyväksi. Paras tapa päästä käsiksi Snort-sääntöön, joka laukaisi hälytyksen, on tallettaa tiedot hälytyksistä MySQL-tietokantaan. Siellä tieto on jäsenneltyä ja helposti haettavissa. Tässä opinnäytetyössä uudeksi talletusmuodoksi on valittu Snorby-tietokanta. Jotta tietoa voidaan tallentaa tietokantaan, tulee Snortin tallettaa tiedot hälytyksistä ensiksi Unified2-tiedostomuotoon. Snortin rinnalle asennetaan Barnyard2-ohjelmisto, joka hoitaa Unified2-tietojen tallettamisen tietokantaan. Koska tietoa hälytyksistä tarvitaan kahteen paikkaan, tulee Snortista ajaa kaksi erillistä instanssia. Toinen välittää tietoa hälytyksistä Ryulle, ja toinen tallettaa datan Unified2-tiedostoon.

Kun tiedot hälytyksistä on talletettu Snorbyn tietokantaan, muokataan Ryun applikaatiota niin, että se hakee tietoa hälytyksistä tietokannasta ja vertaa saamaansa tietoa Snortin rules-sääntötiedostoihin. Näin Ryu saa tietoonsa tarkan Snort-säännön, joka laukaisi hälytyksen. Tämän tiedon pohjalta luodaan uusi vuosääntö Ryun valmiin `_flow_mod`-komponentin avulla, ja välitetään OpenFlow-kytkimelle.

Kun lähetetään uusi vuosääntö kytkimelle eli PacketOut-viesti, tulee olla tiedossa kytkimen Datapath ID (DPID). Alkuperäinen applikaatio saa DPID:n PacketIn-viestin mukana. DPID:n avulla kontrolleri yksilöi jokaisen OpenFlow-kytkimen, ja siten osaa lähettää PacketOut-viestin oikealle laitteelle. Koska PacketOut-viesti lähetetään spontaanisti ilman PacketIn-viestiä, joudutaan DPID-arvot selvittämään muulla keinoin. Ryu sisältää applikaation `topology.api`, jonka avulla voidaan selvittää kytkimien DPID-arvot.

7.3.3 Tarvittavien lisäohjelmien asentaminen

Snorby on WEB-pohjainen, graafisen käyttöliittymän tarjoama MySQL-tietokanta.

Asennetaan ensiksi Snorbyn vaatimat lisäpaketit:

```
sudo apt-get install -y imagemagick apache2 libyaml-dev
libxml2-dev libxslt-dev git ruby1.9.3
```

Koska Snorby asennus on hieman hidas, nopeutetaan sitä hieman ajamalla seuraavat komennot, jotka estävät asennusdokumentaation luomisen:

```
echo "gem: --no-rdoc --no-ri" > ~/.gemrc
sudo sh -c "echo gem: --no-rdoc --no-ri > /etc/gemrc"
```

Asennetaan asennukseen vaadittavat gemit:

```
sudo gem install wkhtmltopdf
sudo gem install bundler
sudo gem install rails
sudo gem install rake --version=0.9.2
```

Snorby on saatavilla GitHub-palvelimelta. Siirrytään kotihakemistoon ja ladataan asennustiedostot, sekä siirretään ladattu kansio WEB-palvelinhakemistoon:

```
cd ~/snort_src/
wget https://github.com/Snorby/snorby/archive/v2.6.2.tar.gz -O
snorby-2.6.2.tar.gz

tar xzvf snorby-2.6.2.tar.gz
sudo cp -r ./snorby-2.6.2/ /var/www/html/snorby/
```

Asennetaan Snorbyn vaatimat lisäosat:

```
cd /var/www/html/snorby
sudo bundle install
```

Käytetään MySQL-esimerkkitiedostoa, ja muokataan sitä vastaamaan nykyistä konfiguraatiota:

```
sudo cp /var/www/html/snorby/config/database.yml.example
/var/www/html/snorby/config/database.yml
```

```
sudo nano /var/www/html/snorby/config/database.yml
```

Muokataan tiedostosta MySQL salasanaa. Tähän lisätään MySQL:n Root-käyttäjän salana kohtaan "mysqlroot". Ohjeessa luodaan omat salasanat Root ja Snorby käyttäjille, joten salasanoja luodessa tulee olla tarkkana! (Katso Kuvio 33)

```

# Snorby Database Configuration
#
# Please set your database password/user below
# NOTE: Indentation is important.
#
snorby: &snorby
  adapter: mysql
  username: root
  password: "mysqlroot" # Example: password: "s3cr3tsauce"
  host: localhost

development:
  database: snorby
  <<: *snorby

test:
  database: snorby
  <<: *snorby

production:
  database: snorby
  <<: *snorby

```

Kuvio 33. Root-käyttäjän lisääminen database.yml-tiedostoon

Luodaan Snorbyn konfiguraatitiedosto, ja muokataan sitä vastaamaan wkhtmlpdf-tiedostoa.

```
sudo cp /var/www/html/snorby/config/snorby_config.yml.example
/var/www/html/snorby/config/snorby_config.yml
```

```
sudo sed -i s/"\usr\local\bin\wkhtmltopdf"/"\usr\bin\wkhtmltopdf"/g /var/www/html/snorby/config/snorby_config.yml
```

Asennetaan Snorby. Asennuksessa saattaa kestää jonkin aikaa. Jammit-virheilmoituksiin ei tarvitse reagoida

```
cd /var/www/html/snorby
sudo bundle exec rake snorby:setup
```

Muokataan MySQL Snorby-tietokantaa. Luodaan uusi käyttäjä (snorby), lisätään sille uusi salasana ja annetaan täydet oikeudet muokata tietokantaa. Otetaan huomioon MySQL:in Root ja Snorby käyttäjien eri salasanat.

```
mysql -u root -p
mysql> create user 'snorby'@'%' IDENTIFIED BY 'PASSWORD123';
mysql> grant all privileges on snorby.* to 'snorby'@'%' with
grant option;

mysql> flush privileges;
mysql> exit
```

Avataan uudelleen database.yml-konfiguraatitiedosto:

```
sudo nano /var/www/html/snorby/config/database.yml
```

Ja muokataan vielä rivejä 8 ja 9. Lisätään äsken luotu käyttäjä snorby, ja sen käyttämä salasana tiedostoon. (Katso Kuvio 34)

```
Snorby Database Configuration
#
# Please set your database password/user below
# NOTE: Indentation is important.
#
snorby: &snorby
  adapter: mysql
  username: snorby
  password: "mysqlsnorby" # Example: password: "s3cr3tsauce"
  host: localhost

development:
  database: snorby
  <<: *snorby

test:
  database: snorby
  <<: *snorby

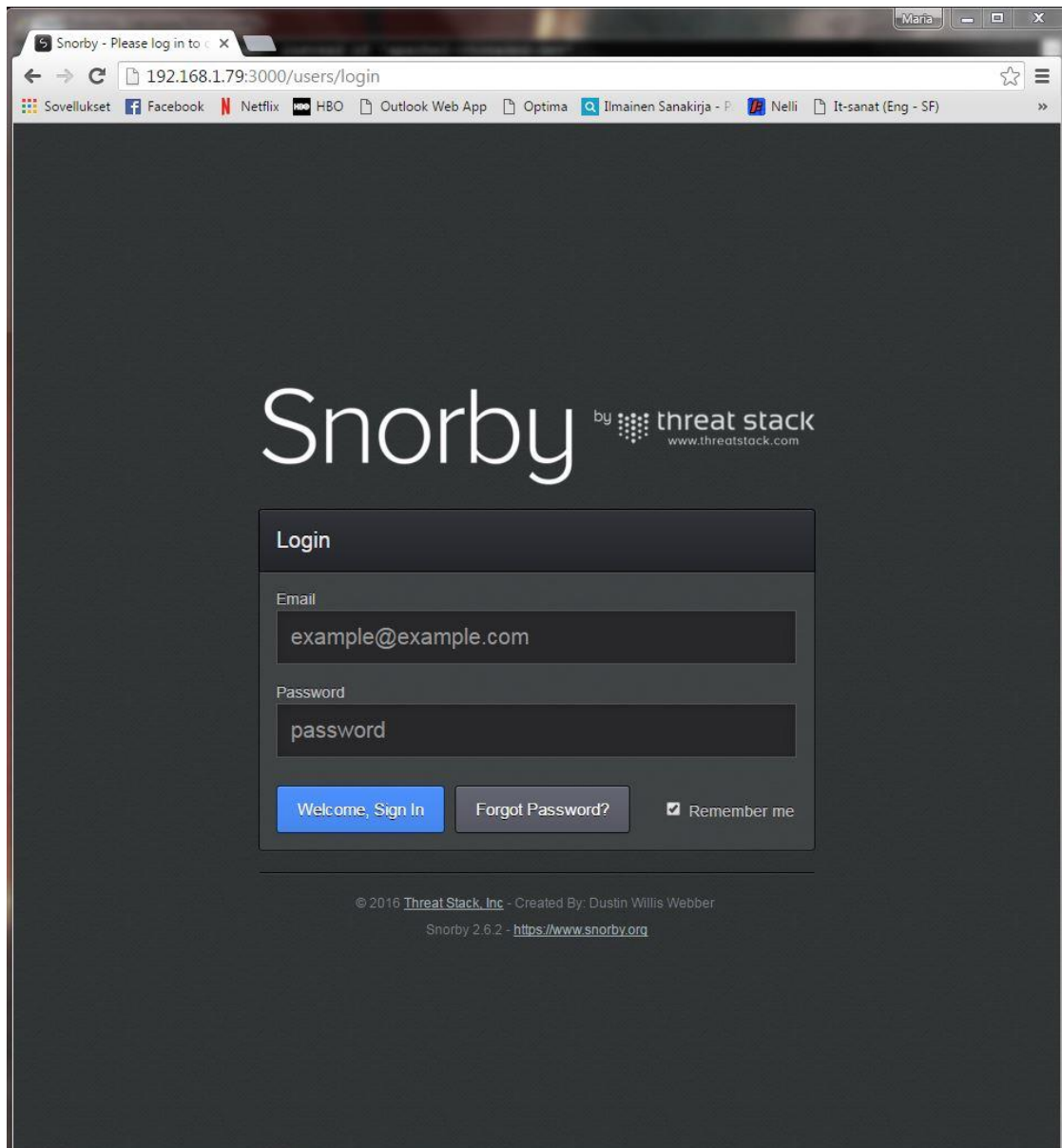
production:
  database: snorby
  <<: *snorby
```

Kuvio 34. Snorby-käyttäjän lisääminen database.yml-tiedostoon

Snorby on valmis testattavaksi. Ajetaan Snorby komennolla:

```
cd /var/www/html/snorby/
sudo bundle exec rails server -e production
```

Snorby käynnistyy ja kuuntelee porttia 3000. Siirrytään WEB-selaimella osoitteeseen "http://<ip_of_snorby_server>:3000". Sivulle aukeaa Snorbyn sisäänkirjautumiskuna (katso Kuvio 35).



Kuvio 35. Snorbyn sisäänkirjautumisikkuna

Käytetään Phusion Passenger-aplikaatiota Apachen tukena Snorbyn käynnistämisalustana. Asennetaan ensiksi vaadittavat lisäosat:

```
sudo apt-get install -y libcurl4-openssl-dev apache2-threaded-dev libaprutil1-dev libapr1-dev
```


Asennetaan Passenger gem ja Apache moduuli:

```
sudo gem install passenger
sudo passenger-install-apache2-module
```

Kun Phusion Passenger-asennusohjelma käynnistyy, valitaan listasta Ruby. Siirry Rubyyn kohdalle ja painetaan välilyöntiä. Lopuksi painetaan Enter-näppäintä, jotta asennus jatkuu.

Asennuksen lopuksi tulee kopioida kuusi ensimmäistä riviä. Kopioitavat rivit näkyvät Kuviossa 36 tummennettuna. Asennusohjelman lopettamiseksi painetaan Enter-näppäintä kahdesti.

```
-----
Almost there!

Please edit your Apache configuration file, and add these lines:

  LoadModule passenger_module /var/lib/gems/1.9.1/gems/passenger-5.0.26/buildout/apache2/mod_passenger.so
  <IfModule mod_passenger.c>
    PassengerRoot /var/lib/gems/1.9.1/gems/passenger-5.0.26
    PassengerDefaultRuby /usr/bin/ruby1.9.1
  </IfModule>

After you restart Apache, you are ready to deploy any number of web
applications on Apache, with a minimum amount of configuration!

Press ENTER when you are done editing.
█
```

Kuvio 36. Phusion Passenger asennusohjelman kopioitavat rivit.

Luodaan tiedosto:

```
sudo nano /etc/apache2/mods-available/passenger.conf
```

Ja lisätään sinne Kuvion 36 ensimmäinen rivi. Kuviossa 37 näkyy kopioitu rivi Passenger.conf-tiedostossa.

```
LoadModule passenger_module /var/lib/gems/1.9.1/gems/passenger-5.0.26/buildout/apache2/mod_passenger.so
```

Kuvio 37. Passenger.conf-tiedosto

Luodaan toinen tiedosto:

```
sudo nano /etc/apache2/mods-available/passenger.load
```

Lisätään loput kaksi riviä kyseiseen tiedostoon. <IfModule> -leimoja ei ole tarpeen lisätä. Katso mallia Kuvioista 38.

```
PassengerRoot /var/lib/gems/1.9.1/gems/passenger-5.0.26
PassengerDefaultRuby /usr/bin/ruby1.9.1
```

Kuvio 38. Passenger.conf-tiedosto

Otetaan Passenger moduuli käyttöön:

```
sudo a2enmod passenger
sudo service apache2 restart
```

ja varmistetaan oheisella komennolla, että se käynnistyy (katso tulostetta)

```
apache2ctl -t -D DUMP_MODULES
```

Luodaan Snorbylle web-sivusto ja muokataan snorby.conf-tiedostoa Kuvion 39 mukaiseksi.

```
sudo nano /etc/apache2/sites-available/snorby.conf
```

```

VirtualHost *:80>
    ServerAdmin webmaster@localhost
    ServerName snorby.sublimerobots.com
    DocumentRoot /var/www/html/snorby/public
    <Directory "/var/www/html/snorby/public">
        AllowOverride all
        Order deny,allow
        Allow from all
        Options -MultiViews
    </Directory>
</VirtualHost>

```

Kuvio 39. Snorby.conf-tiedosto

Poistetaan oletussivu käytöstä, ja otetaan uusi sivu käyttöön.

```

cd /etc/apache2/sites-available/
sudo a2ensite snorby.conf
sudo service apache2 reload
cd /etc/apache2/sites-enabled
sudo a2dissite 000-default
sudo service apache2 reload

```

Snorby tarvitsee vielä tietokannan ylläpitoon palvelun. Luodaan taustapalvelun käynnistys skripti:

```
sudo nano /etc/init/snorby_worker.conf
```

Lisää tiedostoon:

```

description "Snorby Delayed Job"
stop on runlevel [!2345]
start on runlevel [2345]
chdir /var/www/html/snorby
script
exec /usr/bin/ruby script/delayed_job start
end script

```

Muokataan tiedoston oikeuksia niin, että se voidaan suorittaa:

```
sudo chmod +x /etc/init/snorby_worker.conf
initctl list | grep snorby_worker
```

Kirjaudutaan nyt Snorby-tietokantaan WEB-selaimen kautta. Avaa sivusto ”
http://<ip_of_snorby_server>” ja kirjaudu tiedoilla:

E-mail: snorby@snorby.org

Password: snorby

Barnyard2

Barnyard2:n avulla voidaan muuttaa Snortin tallentama Unified2-binääritiedosto luettavaan muotoon ja tallettaa MySQL-tietokantaan.

Asennetaan Barnyard2:n vaatimat lisäosa

```
sudo apt-get install -y mysql-server libmysqlclient-dev mysql-
client autoconf libtool
```

Asennusohjelma pyytää asettamaan MySQL:n Root-käyttäjän salasanan.

Muokataan snort.conf-konfiguraatitiedostoa niin, että se tallettaa hälytykset binäärimuotoisena datana. Lisätään tiedostoon riville 521:

```
output unified2: filename snort.u2, limit 128
```

Ladataan ja asennetaan Barnyard2.

```
cd ~/snort_src wget https://github.com/firnsy/barnyard2/archive/7254c24702392288fe6be948f88afb74040f6dc9.tar.gz -O barn-
yard2-2-1.14-336.tar.gz
```

```
tar zxvf barnyard2-2-1.14-336.tar.gz
```

```
mv barnyard2-7254c24702392288fe6be948f88afb74040f6dc9 barn-
yard2-2-1.14-336 cd barnyard2-2-1.14-336
```

```
autoreconf -fvi -I ./m4
```

Luodaan Barnyard2:lle linkki dumbnet.h kirjastoon.

```
sudo ln -s /usr/include/dumbnet.h /usr/include/dnet.h
sudo ldconfig
```

Rakennetaan asennus prosessi. Riippuen järjestelmän arkkitehtuurista, vain toinen seuraavista komennoista tulee suorittaa.

```
# Valitse toinen seuraavista komennoista
./configure --with-mysql --with-mysql-libraries=/usr/lib/x86_64-linux-gnu

./configure --with-mysql --with-mysql-libraries=/usr/lib/i386-linux-gnu
```

Asennetaan Barnyard2:

```
make
sudo make install
```

Asennuksen jälkeen kopioidaan ja muokataan tiedostoja, joita Barnyard2 tarvitsee:

```
cd ~/snort_src/barnyard2-2-1.14-336
sudo cp etc/barnyard2.conf /etc/ sudo mkdir /var/log/barnyard2
sudo chown snort.snort /var/log/barnyard2
sudo touch /var/log/snort/barnyard2.waldo
sudo chown snort.snort /var/log/snort/barnyard2.waldo
```

Muokataan Barnyard2-konfiguraatitiedostoa, jotta se lisää tapahtumat Snorbyn tietokantaan.

```
sudo nano /etc/snort/barnyard2.conf
```

Lisätään tiedoston loppuun oheinen rivi (lisää salasanan kohdalle Snorby-käyttäjän salasana):

```
output database: log, mysql, user=snorby password=PASSWORD123
dbname=snorby host=localhost sensor_name=sensor1
```

Barnyard voidaan suorittaa seuraavalla komennolla (Argumenttien selitys on Taulukossa 5):

```
sudo barnyard2 -c /etc/snort/barnyard2.conf -d /var/log/snort
-f snort.u2 -w /var/log/snort/barnyard2.waldo
```

Taulukko 5. Barnyard2-komennon argumentit

Argumentti	Selitys
-c	Konfiguraatitiedoston sijainti
-d	Unified2-tiedoston sijainti
-f	Unified2-tiedoston nimi
-w	Checkpoint-tiedosto, jonka avulla Barnyard2 tietää, mitkä tiedot on talletettu jo tietokantaan

7.3.4 Applikaation lähdekoodin kehittäminen

Lähdetään kehittämään simple_switch_snort.py-applikaatiota, jotta integraatio olisi toimivampi, ja otetaan uudet ominaisuudet käyttöön. Uusi, kehitetty applikaatio on nimeltään snort.py, ja se löytyy kokonaisuudessaan tämän opinnäytetyön Liitteestä 2.Liite 1. Simple_switch_snort.py

Kuviossa 40 on tapahtuma, joka suoritetaan aina, kun Ryu saa hälytyksen Snortilta. Ensiksi käytetään Ryu-applikaatiota topology.api:a, jonka avulla selvitetään verkon kytkimien DPID-arvot. Tässä testiversiossa selvitetään ainoastaan kytkimen S1 DPID arvo (Kuviossa 40; sw_sdpid[0]). Kytkimen DPID-arvoa tarvitaan parser-objektin käytössä, sekä lähettäessä uutta vuosääntöä kytkimelle.

Jotta voidaan tehdä kysely Snorby-tietokantaan, tarvitaan jokin yksilöivä tieto, jolla dataa tietokannasta haetaan. Jokaisella IPv4-paketilla on sen otsikkotiedoissa IP-identifikaatio-kenttä. IP ID-luku on juokseva eli se kasvaa aina yhdellä paketin lähettämisen jälkeen. Kentän käyttötarkoitus tulee parhaiten esille TCP-yhteyksissä. IP ID-

kentän avulla voidaan varmistaa, että vastaanottaja kokoaa yhteyden paketit oikeassa järjestyksessä, ja ettei paketteja ole kadonnut lähetyksen aikana.

```
@set_ev_cls(snortlib.EventAlert, MAIN_DISPATCHER)
def _dump_alert(self, ev):
    msg = ev.msg

    # Get all dpid from switches
    sw_dpid = [s.dp.id for s in get_switch(self)]

    # Get datapath by dpid
    datapath = ryu.app.ofctl.api.get_datapath(self, sw_dpid[0])

    # Parse content
    pkt = msg.pkt
    pkt = packet.Packet(array.array('B', pkt))
    _ipv4 = pkt.get_protocol(ipv4.ipv4)
    alertmsg = ''.join(msg.alertmsg)
    ip = _ipv4.src
    id = _ipv4.identification

    print('Received Alert from %s with alert message %s' % (ip, alertmsg))

    self.get_snort_rule(datapath, id, pkt)
```

Kuvio 40. snort.py – dump-alert

Kuviossa 41 on siirretty luokkaan nimeltä "get_snort_rule". Luokassa muodostetaan yhteys Snorby-tietokantaan. Tietokantaan tehdään kaksi MySQL-kyselyä eri tietotauluihin. Ensiksi tehdään kysely tauluun "lphdr", jonne tallentuvat kaikki hälytyksien aiheuttaneiden pakettien IP-otsikkotiedot. Tähän tauluun verrataan EventAlert-tapahtumankäsittelijältä saatua IP ID-arvoa. Applikaatiossa käytetyt tietotaulut on esitetty Liitteessä 3.

Ensiksi etsitään lphdr-tietotaulusta hälytyksestä saatua IP-identifikaation-arvoa. Kun vastaavuus löytyy, otetaan talteen kyseiseltä riviltä cid-arvo. CID-arvo on Snortin automaattisesti määrittämä järjestysnumero hälytyksille. Ensimmäisen hälytyksen cid-arvo on 1, toisen 2, ja niin edelleen. Poimimalla CID-arvon lphdr-taulusta, voidaan tätä lukua jälleen verrata toiseen Snorbyn tietotauluun. Events_with_join-tietotau-

luun tallentuvat tarkemmat tiedot Snortin hälytyksistä. Tietotauluun talletetaan samat tiedot, kuin Alert Fast-talletusmuodossa. CID-arvoa verrataan tähän tietotauluun, jolloin saadaan selville hälytyksen allekirjoitus. Sig_name-kenttä pitää sisällään hälytyksen SID-arvon, joka yksilöin jokaisen Snortin säännön.

Sig_name-kentästä saadaan säännön järjestysnumero eli SID. Tätä tietoa voidaan nyt verrata Snortin sääntötiedostoon. Koska Snort ja Ryu ajetaan eri koneilla, tulee muodostaa FTP-yhteys yhteyden muodostamiseksi. Applikaatio etsii ja lukee tiedoston "ryu.rules", jonne on talletettu testiympäristön säännöt. Applikaatio käy tiedoston läpi rivi riviltä, kunnes löytää rivin, jolla on oikea SID-arvo.


```

def get_snort_rule(self, datapath, id, pkt):
    db = MySQLdb.connect(host="192.168.1.79",      # MySQL host
                        user="snorby",          # username
                        passwd="mysqlsnorby",    # password
                        db="snorby")           # name of the data base

    cur = db.cursor()
    cur.execute("SELECT * FROM iphdr")
    cur.close()

    # Make a query to database
    cur = db.cursor()
    cur.execute("SELECT cid FROM iphdr WHERE ip_id = %s", (id))

    # get data from Snorby
    rows = cur.fetchall()
    for row in rows:
        cid = row
    cur.close()

    # Make another query
    cur = db.cursor()
    cur.execute("SELECT sig_name FROM events_with_join WHERE cid = %s", (cid))

    rows2 = cur.fetchall()
    for row2 in rows2:
        string = ''.join(row2)

    cur.close()

    # Get the clean data from the string
    out = re.compile('(?:.*?)', re.DOTALL | re.IGNORECASE).findall(string)
    if out :
        sid = ''.join(out)

    #Open ftp connection
    ftp = ftplib.FTP('192.168.1.79', 'maria', 'akuankka89')

    #Get the rule file
    ftp.cwd("rules")
    gFile = open("ryu.rules", "wb")
    ftp.retrbinary('RETR ryu.rules', gFile.write)
    gFile.close()
    ftp.quit()

    #Read the file contents
    gFile = open("ryu.rules", "r")
    buff = gFile.read()
    gFile.close()

    #Find the rule, that has correct sid
    with open("ryu.rules") as f:
        for line in f:
            if sid in line:
                line2 = line

    #Make a tuple than data can be compare to alert
    value = tuple(line2.split(' '))
    self.packet_print(value, datapath, pkt)

```

Kuvio 41. Snort.py – get_snort_rule

Applikaatio muuttaa sääntötiedoston rivillä olevat tiedot Pythonin tuple-objektiksi, jolloin tiedot ovat ikään kuin listana. Tällöin rivin arvoja voidaan tutkia helpommin. Tuple-objektissa säännön tiedot ovat Taulukon 6 mukaiset.

Taulukko 6. Tuple-objektin kentät

Tuple kenttä	Arvon tyyppi
value[0]	Säännön toiminta, (Esimerkiksi Alert)
value[1]	Protokollan tyyppi, (ICMP, TCP)
value[2]	Lähettäjän IP-osoite
value[3]	Lähettäjän portin numero
value[4]	Säännön suunta
value[5]	Vastaanottajan IP-osoite
value[6]	Vastaanottajan portti

Kuviossa 42 esitellään applikaation seuraava luokka. Packet_print-luokassa tutkitaan Snortin-sääntöä, joka laukaisi hälytyksen. Ensiksi tutkitaan käytetty protokolla. Applikaatio ottaa huomioon ainoastaan IP-paketin ICMP ja TCP protokollat. Tiedot talletetaan Pythonin dictionary-objektiin. Luokassa käydään myös läpi lähettäjän ja vastaanottajan IP-osoitteet sekä porttinumerot. Jos tuple-objektin arvossa on muuta dataa kuin "any", niin tieto talletetaan dictionary-objektiin.

Dictionary-objektissa olevia tietoja käytetään uuden vuosäännön vastaavuuden määrittämiseen. Objekti annetaan OFPMatch.parser-funktiolle käsiteltäväksi, joka luo annetuista tiedoista OpenFlow-vuosäännön Match-kenttä. Match-kenttä vältetään edelleen send_flow_mod-luokalle.

```

def packet_print(self, value, datapath, pkt):
    datapath = datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # Create dictionary of existing values

    if value[1] in ('icmp'):
        ip_proto=1

    if value[1] in ('tcp'):
        ip_proto=6

    dict = {'eth_type': 2048};
    dict['ip_proto'] = ip_proto;

    if value[2] != ('any'):
        ipv4_src = value[2]
        dict['ipv4_src'] = ipv4_src;

    if value[3] != ('any'):
        src_port = value[3]
        dict['tcp_src'] = int(src_port);

    if value[5] != ('any'):
        ipv4_dst = value[5]
        dict['ipv4_dst'] = ipv4_dst;

    if value[6] != ('any'):
        dst_port = value[6]
        dict['tcp_dst'] = int(dst_port);

    # Use data of dictionary in OFPMatch parser
    match = dict
    match = parser.OFPMatch(**match)

    self.send_flow_mod(datapath, match)

```

Kuvio 42. Snort.py – packet_print

Kuviossa 43 on esitetty Send_flow_mod-luokka. Sen tehtävä on luoda vuosääntö loppuun Packet_print-luokalta saadun Match-kentän avulla. Luokassa on ennalta määritetty säännön prioriteetti sekä toiminto säännölle. Säännön toiminta on OFPActionOutput (0) eli välitä paketti porttiin nolla. Toisin sanoen vuosääntöä vastaavat paketit pudotetaan. Luokka välittää lopuksi luodun vuosäännön kytkimelle.

```

# Create new flow to block packet next time
def send_flow_mod(self, datapath, match):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    cookie = cookie_mask = 0
    table_id = 0
    idle_timeout = hard_timeout = 0
    priority = 32768
    buffer_id = ofp.OFP_NO_BUFFER
    actions = [ofp_parser.OFPActionOutput(0)]
    inst = [ofp_parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS,
                                             actions)]
    req = ofp_parser.OFPFlowMod(datapath, cookie, cookie_mask,
                                table_id, ofp.OFPFC_ADD,
                                idle_timeout, hard_timeout,
                                priority, buffer_id,
                                ofp.OFPP_ANY, ofp.OFPG_ANY,
                                ofp.OFPFF_SEND_FLOW_REM,
                                match, inst)

    datapath.send_msg(req)

```

Kuvio 43. Snort.py – Send_flow_mod-luokka

7.4 Toinen testi

Testataan käytännössä, kuinka uusi applikaatio, snort.py toimii. Muokataan sääntö-tiedostoa `/etc/snort/rules/ryu.rules` Snort-virtuaalikoneella. Muokataan sitä niin, että Snort aiheuttaa hälytyksen, kun se havaitsee ICMP-paketin Host 1:ltä Host 2:lle, sekä TCP-paketin Host 1:ltä Host 2:lle porttiin 80. Esimerkki on Kuviossa 44.

```

GNU nano 2.2.6                               File: /etc/snort/rules/ryu.rules               Modified
alert icmp 10.0.0.1 any -> 10.0.0.2 any (msg:"Ping message discovered"; sid:1000004; rev:1;)
alert tcp 10.0.0.1 any -> 10.0.0.2 80 (msg:"HTTP-connection to Host 2"; sid: 1000003;rev:1;)

```

Kuvio 44. Toisen testin säännöt.

Käynnistetään Ryu Snort.py-applikaatio Ryu-virtuaalikoneella:

```

cd ryu/ryu/app
sudo ryu-manager snort.py

```

Ajetaan Mininet Snort-virtuaalikoneella samalla komennolla, kuin ensimmäisessä testissä:

```
sudo mn --topo linear,2 --switch ovsk,protocols=OpenFlow13
--controller remote,ip=191.168.1.80,port=6633
```

Luodaan Host 2:lle HTTP-palvelin komennolla:

```
h2 python -m SimpleHTTPServer 80 &
```

Käynnistetään Snort terminaalissa 2 (Alt + F2):

```
sudo snort -i s1-eth2 -A unsock -l /tmp
-c /etc/snort/snort.conf
```

Käynnistetään terminaalissa 3 toinen Snort-instanssi, joka tallettaa hälytykset Unified2-tiedostomuotoon:

```
sudo snort -i s1-eth2 -c /etc/snort/snort.conf
```

Käynnistetään Barnyard2-applikaatio terminaalissa 4, joka tallettaa Unified2-tiedot Snorby-tietokantaan:

```
sudo barnyard2 -c /etc/snort/barnyard2.conf -d
/var/log/snort/ -f snort.u2
-w /var/log/snort/barnyard2.waldo
```

Lopuksi käynnistetään vielä Pigrelay-applikaatio, joka kuuntelee Snortilta tulevia hälytyksiä, ja välittää ne Ryu-kontrollerille:

```
cd /etc/snort/pigrelay
sudo python pigrelay.py
```

Lähetetään ICMP ECHO-viesti Host 1:ltä Host 2:lle

```
h1 ping h2
```

Host 1 lähettää tasaisena virtana ICMP-paketteja host 2:lle. Kun katsotaan Ryu konsolia, nähdään siinä Kuvion 45 mukainen viesti. Kuten kuvioista näkyy, Host 1 ehtii lähettää kaksi ICMP-pakettia ennen kuin Ryu uusi vuosääntö kirjautuu kytkimen S1 vuotauluun.

```
[snort][INFO] Connected with 192.168.1.79
Received Alert from 10.0.0.1 with alert message Ping message discovered
Received Alert from 10.0.0.1 with alert message Ping message discovered
```

Kuvio 45. Hälytys viesti Ryu-konsolilla

Lähetetään vielä HTTP-pyyntö Host 1:ltä Host 2:n WEB-serverille komennolla:

```
h1 wget -O - h2
```

Kuviosta 46 nähdään uusi hälytys Ryu-konsolilla. Ryu luo uuden vuosäännön, kun saa hälytyksen Snortilta. Kun HTTP-pyyntö ajetaan uudestaan Host 1:ltä, pyynnön lähetys epäonnistuu uuden vuosäännön vuoksi.

```
[snort][INFO] Connected with 192.168.1.79
Received Alert from 10.0.0.1 with alert message Ping message discovered
Received Alert from 10.0.0.1 with alert message Ping message discovered
Received Alert from 10.0.0.1 with alert message HTTP-connection to Host 2
```

Kuvio 46. HTTP-hälytys Ryu-konsolilla

Vuotaulu on nyt Kuvion 47 mukainen. Kuvan ensimmäisessä vuosäännössä määritetään ICMP-sääntö. Viestintyyppi on icmp, paketin lähettäjän IP-osoite (nw_src) on 10.0.0.1 ja vastaanottajan osoite (nw_dst) on 10.0.0.2. Output-portti on 0 eli paketti

pudotetaan. Toinen vuosääntö koskee TCP-yhteydenottoja porttiin 80 osoitteesta 10.0.0.1 osoitteeseen 10.0.0.2.

```

root@ubuntu:/home# ovs-ofctl dump-flows s1 -O openflow13
OFFST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=130.953s, table=0, n_packets=71, n_bytes=6958, send_flow_rem icmp,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:0
 cookie=0x0, duration=52.865s, table=0, n_packets=0, n_bytes=0, send_flow_rem tcp,nw_src=10.0.0.1,nw_dst=10.0.0.2,tp_dst=80 actions=output:0
 cookie=0x0, duration=133.178s, table=0, n_packets=17, n_bytes=2295, priority=1,in_port=2,dl_dst=4a:33:02:72:4b:30 actions=output:1,output:3
 cookie=0x0, duration=133.169s, table=0, n_packets=16, n_bytes=1106, priority=1,in_port=1,dl_dst=ae:ba:32:6c:0d:86 actions=output:2,output:3
 cookie=0x0, duration=141.871s, table=0, n_packets=6, n_bytes=392, priority=0 actions=CONTROLLER:65535

```

Kuvio 47. Ryu on luonut uudet vuosäännöt kytkimen S1 vuotauluun.

Koska Snort-sääntö estää ainoastaan TCP-yhteydenotot osoitteesta 10.0.0.1 osoitteeseen 10.0.0.2, voidaan Host 2:lta luoda onnistuneesti TCP-yhteyksiä Host 1:sen porttiin 80. Toisaalta, myös ICMP-viestin lähettäminen Host 2:lta Host 1:lle onnistuu, mutta koska kytkin pudottaa Host 1:ltä tulevan vastausviestin, ping-testi epäonnistuu.

7.5 Kolmas testi

Toteutetaan Snort.py-aplikaatiolle vielä rasiustesti, ja testataan kuinka hyvin se suoriutuu oikeasta DoS-hyökkäyksestä. Luodaan uusi virtuaalikone, johon asennetaan Ubuntu 14.04-Desktop, joka toimii Mininet-verkkoon päätelaitteena, Host 3:na. Koska Mininet-verkossa ei ole reititintä, on Host 3-laitteelle annettu IP-osoite samasta osoiteavaruudesta, kuin muille Mininetin päätelaitteille. Yhdistetään Host 3 kytkimeen S2 komennolla:

```
sudo ovs-ofctl add-port s2 p3p2
```

Host 3:lle asennetaan Hping3-ohjelma, jolla voidaan luoda aitoja palvelunestohyökkäyksiä. Hping3-ohjelman voi asentaa APT-asennustyökalun avulla.

Asennetaan Hping3:

```
sudo apt-get install hping3
```

Luodaan Mininet-verkon Host 1:lle WEB-palvelin, joka toimii hyökkäyskohteena:

```
h1 python -m SimpleHTTPServer 80 &
```

Lisätään vielä uusi sääntö tiedostoon ryu.rules, joka hälyttää TCP-paketeista, jotka tulevat IP-osoitteesta 10.0.0.12 (Host 3-laitteen IP-osoite)

```
alert tcp 10.0.0.12 any -> 10.0.0.1 80 (msg: "DoS attack";
sid: 1000006; rev:1;)
```

Muokataan Ryu-koneella Snort.py-aplikaatiota niin, että se lähettää uudet vuosäähäntöt kytkimelle S2. Katso Kuviosta 48 oikea konfiguraatio.

```
# Get datapath by dpid
datapath = ryu.app.ofctl.api.get_datapath(self, sw_dpid[1])
```

Kuvio 48. Snort.py konfiguraation muutos

Lisätään verkkoon kaksi tcpdump-instanssia, jotka tarkkailevat verkon liikennettä. Sijoitetaan toinen tcpdump-instanssi kytkimen S2 rajapintaan p3p2. Tästä instanssista nähdään, kuinka monta TCP-pakettia Hping3 lähettää Mininet-verkkoon. Sijoitetaan toinen tcpdump-instanssi kytkimen S1-rajapintaan s1-eth1. Tästä rajapinnasta voidaan nähdä, kuinka monta Hping3:n lähettämää TCP-pakettia saapuu Host 1:lle. Tcpcdump instanssi voidaan ajaa vapaassa terminaali-ikkunassa. Mikäli vapaita terminaleja ei ole, voidaan muodostaa SSH-yhteys Snort koneeseen virtuaalikoneiden isäntälaitteesta.

Aloitetaan tcpdump-instanssit kytkimillä S1 ja S2 (molemmat instanssit vaativat omat ikkunat):

```
tcpdump -I s1-eth1 -n -w host1.cap
tcpdump -I p3p2 -n -w incoming.cap
```


Käynnistetään loput ohjelmat sekä instanssit, kuten testissä kaksi. Ajetaan Hping3 Host 3:lta. Hping-komennossa määritetään interval, eli pakettien lähtöviive argumentilla -i. Kuviossa 49 viiveeksi on määritetty 10 000 mikrosekuntia. Toisin sanoen Hping3 lähettää kymmenen pakettia sekunnissa. -S argumentilla tarkoitetaan SYN-FLOOD-hyökkäystä. -p argumentilla määritetään kohdeportin numero 80, ja 10.0.0.1 on IP-osoite, johon hyökkäys kohdistetaan. Lopuksi argumentilla -I eth1 määritetään rajapinta, josta hyökkäys suunnataan ulos. Tämä ei ole tarpeellinen, mikäli käytössä on vain yksi rajapinta.

A terminal window with a dark background and light text. The title bar shows 'maria@ubuntu: ~'. The command prompt is 'maria@ubuntu:~\$' followed by the command 'sudo hping3 -i u10000 -S -p 80 10.0.0.1 -I eth1'. A white cursor is at the end of the command.

Kuvio 49. Hping3

Aloitetaan hyökkäys, ja annetaan sen kestää noin 5-7 sekuntia. Keskeytetään hyökkäys Ctrl + C näppäinyhdistelmällä.

7.5.1 Tulosten analysointi

Tähän kappaleeseen on koottuna kolmannen testin tuloksia. Testi ajettiin neljällä eri paketin lähetysnopeudella, jotta nähdään, kuinka hyvin Ryu kykenee reagoimaan aggressiiviseenkin hyökkäykseen. Jokaisella nopeudella testi ajettiin useampaan kertaan keskiarvotilastojen saamiseksi. Tulokset ovat kokonaisuudessaan Liitteessä 4.

Taulukoiden ensimmäinen sarake (Hyökkäys) kertoo, kuinka monta TCP-pakettia Hping3 lähetti Mininet-verkkoon. Tulokset on saatu, kun tcpdump-instanssin data on luettu Wireshark-ohjelmalla. Koska hyökkäyksien kestot pyrittiin pitämään mahdollisimman tasaisina, ovat lähetettyjen pakettien määrät samaa suuruusluokkaa. Muutama piikki pakettien määrässä johtuvat Hping3:sen lähettämistä RST-lipulla varuste-

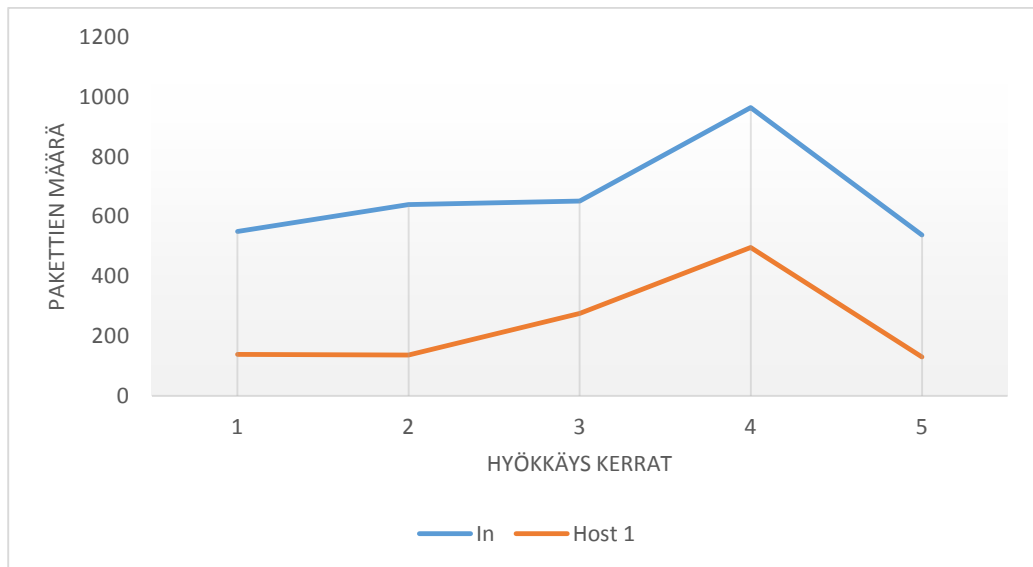
tuista paketeista. Kun Hping3 lähettää TCP RST-paketin, on todellinen pakettien lähetysnopeus suurempi, kuin pelkkien TCP SYN-pakettien. RST-lipuilla varustetuilla paketeilla Hping3 pyrkii resetoimaan aikaisemmin luotuja yhteyksiä, jotta pystyy luomaan uusia yhteyksiä kohteeseensa.

Taulukkojen toinen sarake (Host 1) kuvastaa, kuinka monta Hping3:sen lähettämistä paketeista saavutti määränpänsä eli WEB-palvelimen Host 1:sen. Tämä tieto perustuu ensimmäisen sarakkeen tavoin tcpdump-instanssista saatuun dataan. Kolmas sarake (OpenFlow-drop) kertoo kuinka monta pakettia Ryu onnistui kaiken kaikkiaan pudottamaan. Pudottettujen pakettien määrän näkee kytkimen vuotaulun tilastotiedoista (Esimerkki aikaisemmassa Kuviossa 47, n_packets)

Taulukossa 7 hyökkäys Host 1:lle on toteutettu maltillisella nopeudella 100 pkt/sek. Kuviossa 50 nähdään sama tulos graafisena käyränä. Sininen käyrä kuvastaa Mininet-verkkoon lähetettyjen pakettien määrää, ja oranssi käyrä Host 1 vastaanottamien pakettien määrää. Kuviossa nähdään, että sininen ja oranssi käyrä seuraavat toisiaan tasanisesti. Piikki sinisessä käyrässä aiheuttaa piikin myös oranssiin käyrään. Piikki johtuu Hping3 lähettämistä RST-lipulla varustetuista TCP-paketeista. Piikin kohdalla todellinen pakettien lähetysnopeus on ollut suurempi, kuin 10 pkt/sek, jonka vuoksi pakettejakin on päässyt Host 1:lle enemmän.

Taulukko 7. Hping3-hyökkäys nopeudella 100 pkt/sek

Hyökkäys	Host 1	OpenFlow -drop
550	139	451
640	137	547
652	276	502
964	496	512
538	131	434

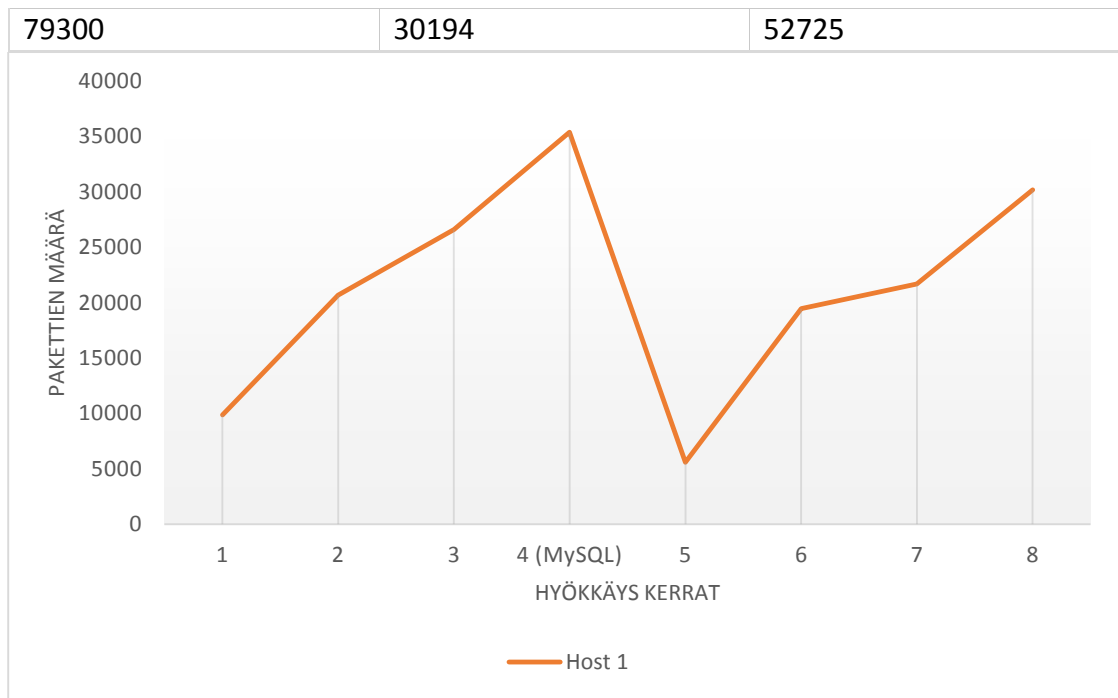


Kuvio 50. Hping3-hylkkäys nopeudella 100 pkt/sek

Taulukossa 8 paketteja lähetetään jo Mininet suorituskyvyn rajoilla 10 000 pkt/sek. Oranssin kaksoisviivan kohdalla on suoritettu MySQL-taulun tyhjennys. Graafisesta esityksessä (Kuvio 51) on esitetty ainoastaan Host 1:lle päässeiden pakettien määrä tuloksen hahmottamisen helpottamiseksi. Kuvio 51Kuvio 51. nähdään selvästi, kuinka Host 1:lle päässeiden pakettien määrä kasvaa jokaisella hyökkäys kerralla, kunnes romahtaa, ja jatkaa sitten uudelleen kasvuaan. Käyrän kohoaminen johtuu MySQL-taulun täydentymisestä, joka hidastaa Ryun tekemien kyselyjen suorittamista. Käyrän laskun kohdalla on tehty MySQL-taulun tyhjennys, jonka jälkeen MySQL-taulu alkaa uudelleen nopeasti täydentyä ja uudelleen hidastamaan Ryun toimintaa.

Taulukko 8. Hping3-hyökkäys nopeudella 10 000 pkt/sek

Hyökkäys	Host 1	OpenFlow-drop
82805	9876	74726
87107	20682	72014
80175	26607	54356
95424	35402	65855
138795	5577	123777
76324	19456	59084
74490	21689	55297



Kuvio 51. Hping3-hyökkäys nopeudella 10 000 pkt/sek

7.5.2 Tulosten yhteenveto

Ensimmäinen testi Snort.py-aplikaation parissa osoitti, että Ryu kykenee luomaan uuden vuosäännön, saadessaan hälytyksen Snortilta. Toisessa testissä testattiin Ryun toimintaa oikeassa DoS-hyökkäyksessä. Hyökkäyksiä ajettiin useampi eri hyökkäysnopeudella keskivertotilastojen saamiseksi. Jokaisella hyökkäys kerralla Ryu kykeni katkaisemaan hyökkäyksen. Kahdella ensimmäisellä hyökkäys kerralla, kun pakettien lähetysnopeus oli vain 100 pkt/sek ja 1000 pkt/sek, Ryu kykeni katkaisemaan hyökkäyksen melko nopeasti, ja vahinko jäi minimaaliseksi. Kuitenkin nopeilla, todellisuutta vastaavilla hyökkäysnopeuksilla WEB-palvelimelle ehtii tulemaan useampi kymmenen tuhatta yhteyden avauspyyntöä. Jo pelkästään 10 000 yhteyden avauspyyntöä palvelimelle on paljon, ja riippuen palvelimen suorituskyvystä, aiheuttaa se ainakin hetkellisen palveluneston todellisille käyttäjille. Koska Ryu kykeni aina lopulta katkaisemaan hyökkäyksen, jää palvelunesto voimaan ainoastaan noin sekunniksi, jolloin sillä ei ole palvelun todellisille käyttäjien kannalta merkitystä.

Kuitenkin todellisissa palvelunestohyökkäyksissä, hyökkääjä käyttää yleensä useampaa konetta ja vaihtelee sattumanvaraisesti IP-osoitteitaan. Tällöin Ryu tulee torjua useampi hyökkäys. Koska Ryu reagoi saamiinsa hälytyksiin noin sekunnin viiveellä, ja jos Ryu saa 10 000 hälytystä, menee sillä 10 000 sekuntia vastata näihin hälytyksiin. Kun Ryu luo yhä edelleen uusia vuosääntöjä jo vanhaksi tulleet hälytykseen, ei se kykene vastaamaan uusiin hälytyksiin tai viesteihin ja on näin ollen muiden laitteiden saavuttamattomissa. Koska MySQL-tietotaulut täyttyvät nopeasti, ja kyselyn tekeminen tietokantaan hidastuu, hidastuu Ryu myös entisestään. Tällöin hyökkääjä onnistuu tehtävässään ja onnistuu saamaan aikaiseksi todellisen palveluneston.

8 Pohdinta

Heti tämän työn toteutusvaiheen alussa, ensimmäisessä testissä huomattiin, että Ryu ja Snortin välinen integraatio on hyvin puutteellinen. Ryu sai tiedon Snortilta haitallisesta liikenteestä, mutta Ryu ei reagoinut hälytykseen toivotulla tavalla. Työn uudeksi tavoitteeksi otettiin siten kehittää integraatiota niin, että sen olemassa olo voidaan todentaa. Toteutusta kehittäessä huomattiin kuitenkin useita ongelmia.

8.1 Integraation ongelmat

Yksi merkittävä integraation ongelma on OpenFlow-vastaavuuksien suppeus. Snorttiin voidaan luoda sääntöjä hyvin tarkasti kohdistumaan juuri tietynlaiseen liikenteeseen. Snort kykenee tutkimaan pakettien kuormaosaa sekä OSI-mallin L5-L7 tason dataa. OpenFlow protokollaan voidaan määrittää sääntöjä pääasiassa ainoastaan L2 ja L3 tason protokolliin. Taulukko 9 listaa OpenFlow 1.3 kaikki mahdolliset Match-kentät. Toisin sanoen, jos Snort tekee hälytyksen paketista kuormaosan perusteella, saattaa Ryu pudottaa myös sallittuja paketteja. Esimerkiksi Snort tekee hälytyksen sisäverkon laitteesta, joka lähettää TCP-paketin WEB-palvelimelle, jonka kuormaosassa on sana "GET". Tarkoitus on pudottaa paketit, jotka sisältävät sanan "GET", mutta Ryu luokin vuosäännön, joka pudottaa kaikki TCP-paketit, joilla on säännön mukaiset IP-osoitteet.

Taulukko 9. OpenFlow 1.3 Match-kentät (Ryu Project Team 2014)

Kentän nimi	Selvitys
in_port	Vastaanottavan portin numero
in_phy_port	Vastaanottavan portin fyysinen numero
metadata	Metadata, taulujen välinen tiedonsiirto
eth_dst	Vastaanottajan MAC-osoite
eth_src	Lähettäjän MAC-osoite
eth_type	Kehystyyppi
vlan_vid	VLAN ID
vlan_pcp	VLAN PCP
ip_dscp	IP DSCP
ip_ecn	IP ECN
ip_proto	IP-paketin protokolla tyyppi
ipv4_src	Lähettäjän IPv4-osoite
ipv4_dst	Vastaanottajan IPv4-osoite
tcp_src	Lähettäjän TCP-portin numero
tcp_dst	Vastaanottajan TCP-portin numero
udp_src	Lähettäjän UDP-portin numero
udp_dst	Vastaanottajan UDP-portin numero
sctp_src	Lähettäjän SCTP-portin numero
sctp_dst	Vastaanottajan SCTP-portin numero
icmpv4_type	ICMP tyyppi
icmpv4_code	ICMP koodi
arp_op	ARP-viestin Op-koodi
arp_spa	ARP-lähettäjän IP-osoite
arp_tpa	ARP-kohteen IP-osoite
arp_sha	ARP-viestin lähettäjän MAC-osoite
arp_tha	ARP-viestin kohteen MAC-osoite
ipv6_src	Lähettäjän IPv6-osoite
ipv6_dst	Vastaanottajan IPv6-osoite
ipv6_flabel	IPv5 flow label
icmpv6_type	ICMPv6 tyyppi
icmpv6_code	ICMPv6 koodi
ipv6_nd_target	IPv6 naapurikyselyn kohde osoite
ipv6_nd_sll	Naapurikyselyn lähettäjän IPv6 link-layer osoite
ipv6_nd_tll	Naapurikyselyn vastaanottajan IPv6 link-layer osoite
mpls_label	MPLS label
mpls_tc	MPLS traffic class (TC)
mpls_bos	MPLS BoS bit
pbb_isid	I-SID of 802.1ah PBB
tunnel_id	Metadata about logical port
ipv6_exthdr	Pseudo-field of extension header of IPv6

Toinen merkittävä ongelma on integraation viive. Jos HTTP-palvelimeen kohdistuu DoS-hyökkäys, ehtii hyökkääjä lähettämään kymmeniä tuhansia HTTP-avauspyyntöjä, ennen kuin Ryu saa luotua kytkimelle vuosäännön, joka estää uudet avauspyynnöt. Vaikka Ryu lopulta onnistuukin keskeyttämään hyökkäyksen, jo 10 000 yhteyden avauspyynnön avulla voidaan luoda ainakin hetkellinen palvelunesto todellisille käyttäjille. Vaikka palvelin on vapaa vastaamaan uusiin HTTP-pyyntöihin, jatkaa Ryu reagointiaan vanhoihin hälytyksiin. Tällöin se ei kykene reagoimaan esimerkiksi verkon kytkimiin PacketIn-paketteihin. Toisin sanoen Ryu on muiden laitteiden saavuttamattomissa.

Viiveettä integraatiossa synnyttävät useat komponentit. Ennen kuin tieto välittyy Snortilta Pigrelay-applikaation kautta Ryulle, ja siitä edelleen kytkimille, puhutaan useasta sadasta millisekunnista, joka aggressiivisessa hyökkäyksessä on merkittävä. Kuitenkin merkittävimmän hitauden integraatioon tuo MySQL. Aggressiivisessa hyökkäyksessä, jossa liikennettä tulee paljon useasta lähteestä, MySQL-taulut täydentyvät nopeasti. Kun tietotauluissa alkaa olemaan lähemmäs miljoona riviä dataa, on MySQL-kyselyn toteuttaminen tuskaisen hidasta.

8.2 Tulosten luotettavuus

Työn testiympäristönä toimi virtuaalinen lähiverkko, Mininet, jossa ajettiin OpenvSwitch-ohjelmiston tuella OpenFlow-kytkimiä. Ympäristö toimii erinomaisesti testitarkoitukseen, mutta luotettavampia tuloksia testeistä saataisiin oikeilla, fyysisillä laitteilla aidossa SDN-verkossa. Kaikki integraation komponentit toteutettiin yhdellä isäntälaitteella, jonka suorituskyky on rajallinen.

Testien suorittaminen paremmin suunnitellussa ympäristössä toisi todennäköisesti parempia testituloksia. Jos integraatio toteutettaisiin oikeassa ympäristössä suorituskyvyltään tehokkailla laitteilla, Ryu reagoisi nopeammin hyökkäyksiin. Suuressa liikenne määrässä Snort instanssit ja MySQL-kyselyt vievät hurjasti tehoa laitteistolta, ja kaikkien ohjelmien ajaminen samalla laitteella vie prosessorin ääri rajoille. Hajautta-

malla ohjelmat ja instanssit omille koneilleen saataisiin integraatio toimimaan luotettavammin. Kuitenkaan, ei voida varmuudella sanoa, riittääkö mikään prosessoritehon määrä paikkaamaan itse integraation hitautta.

Jo pelkästään tämän opinnäytetyön testitulosten perusteella voidaan todentaa, että integraatio Ryu-kontrollerin ja Snort-tunkeilijan estojärjestelmän välillä on olemassa, ja se edistää SDN-verkon tietoturvaa. Integraatio ei kuitenkaan tällaisenaan tuo paljoa hyötyä aidolle SDN-verkolle, vaan integraatiota tulee edelleen kehittää. Jos integraation toimintaa ja suorituskykyä saadaan parannettua jatkokehityksessä, löytyisille varmasti paikka SDN-verkosta ja saatu hyöty tietoturvallisuuden suhteen olisi merkittävä.

8.3 Applikaation jatkokehitys

Jatkokehittämisen kannalta tulee ottaa huomioon OpenFlow versioiden 1.4 ja 1.5 tuomat uudet mahdollisuudet. OpenFlow toimii pääasiassa OSI-mallin tasoilla L2 ja L3. OpenFlow versio 1.4 ei tuo mukanaan paljoa uutta, mutta versio 1.5 tuo mukanaan tuen L4-tason tarkasteluun. Se pystyy havainnoimaan TCP-yhteyden avaamisen ja sulkemisen tutkimalla TCP-paketin SYN, ACK ja FIN-lippuja. (L4-L7 Service Function Chaining Solution Architecture 2015.)

OpenFlow versioon 1.5 tutkitaan myös mahdollisuutta tuoda mukaan OSI-mallin tasojen L5-L7 luokittelu. Kuitenkin salauksen ja pakettien purkaminen ovat tuoneet ominaisuuden toteuttamisen haastavaksi. (Openflow Switch Specificatio Version 1.5.1 2015.)

Snort.py-applikaatio perustuu siihen, että Ryu luo uudet vuosäännöt kytkimille Snortin sääntöjen perusteella. Täten Ryun pitää selvittää hälytyksen laukaissut sääntö, joten ohjelmaan on lisätty kaksi MySQL-kyselyä sekä FTP-tiedoston haku. Nämä kaksi seikkaa hidastavat ohjelmaa merkittävästi, ja ohjelma onkin noin 50 % hitaampi kuin alkuperäinen ohjelma. Integraation toimintaa tulee muokata niin, että jos se saa useita hälytystä samasta hyökkäyksestä, se reagoisi vaan ensimmäiseen, eikä jää luomaan uutta vuosääntöä jokaiselle hälytykselle. Snortilla on ominaisuus, jossa se tunnistaa DoS-hyökkäyksen, ja luo sen perusteella ainoastaan yhden hälytyksen. Tämän toiminnon käyttöönottoa suositellaan jatkokehityksessä.

MySQL aiheuttaa hitauden lisäksi myös muita ongelmia. Vaikka Barnyard2 on nopea kirjaamaan dataa tietokantaan, tekee se sen pienellä viiveellä. Tämän vuoksi myös Ryu saattaa keskeytyä virheeseen, koska se ei löydä tarvitsemaansa dataa tietotaulusta. Koska MySQL on yksisäikeinen ohjelma, sallii se vain yhden avoimen yhteyden tietokantaan. Kaksi MySQL-kyselyä synnyttää kaksi yhteyttä tietokantaan, joka aiheuttaa ohjelmaan myös virheitä. Jatkokehityksessä applikaatio kannattaa toteuttaa niin, että ohjelma ei keskeydy MySQL-virheeseen.

Koska Snortista ajetaan kaksi instanssia, toinen lähettää hälytyksen Ryulle ja toinen tallettaa datan Unified2-tiedostomuotoon, joudutaan jokainen paketti käymään myös kahdesti läpi. Tämä kasvattaa Snort-koneen prosessorin kuormaa. Jos verkossa on paljon liikennettä, on ratkaisuna ajaa molemmat instanssit eri koneilla. Toinen vaihtoehto on korvata MySQL-talletus esimerkiksi lokikirjauksella. Lokikirjaus voidaan suorittaa samaan aikaan, kun dataa lähetetään Network Socketin kautta toiselle järjestelmälle. Lokikirjauksesta on kuitenkin vaikeampi hakea dataa, koska MySQL-tietotauluissa data on jäsenneiltyä. MySQL-tietokanta on myös hyödyllinen silloin, kun käyttäjä haluaa saada jälikäteen tilastotietoja hyökkäyksistä.

Sääntöjen kirjaaminen manuaalisesti vastaamaan kaikki mahdollisia hyökkäysskenaarioita on mahdotonta. Snortin preprocessors-lisäosien avulla Snort kykenee havaitsemaan hyökkäyksiä itsenäisesti ja luomaan hyökkäysten pohjalta dynaamisesti uusia sääntöjä. Muokkaamalla preprocessorin toimintaa niin, että Ryu kykenee käyttämään hyväksi dynaamisia sääntöjä, voidaan suojautua yllättäviltäkin hyökkäyksiltä. FTP-tiedostohaussa on myös puutteensa. Tällä hetkellä FTP etsii vain yhtä tiettyä tiedostoa, ja käy sen läpi etsien vastaavuutta Snort-ID-lukuun. Jatkokehityksessä FTP tulisi ohjelmoida niin, että se etsii vastaavuutta useammasta tiedostosta. Tämä aiheuttaa kuitenkin yhä enemmän viivettä ohjelmaan, jonka vuoksi FTP kannattaa ennemmin korvata jatkokehityksessä joillain toisella tekniikalla.

Applikaatio on kehitetty niin, että se toimii vain testiympäristössä. Siksi Ryu lähettää uuden vuosäännön vain yhdelle, ennalta määrätylle kytkimelle. Monimutkaisemmassa topologiassa tämä asetus ei enää toimi, vaan ohjelmaa tulee muokata niin, että vuosääntö lähetetään dynaamisesti usealle kytkimelle. Jatkokehityksessä kannattaa myös ottaa huomioon kontrollerin ja kytkimien välisen liikenteen salaus. Ryu sisältää TLS/SSL salauksen, jonka käyttöönotto kasvattaa tietoverkon tietoturvaa entisestään.

Lähteet

- Al Braiki, A., Trabelsi, Z., Hayawi, K., Sujith, M. 2013. Network Attacks and Defenses: A Hands-on Approach. E-kirja. Auerbach Publications.
- Alder, R. 2004. Snort 2.1 Intrusion Detection, Second Edition. E-kirja. Syngress publishing.
- Biles, S., Orebaugh, A., Babbin, J. 2005. Snort Cookbook. E-kirja. O'Reilly Media.
- Black, C., Göransson, P. 2014. Software Defined Networks: A Comprehensive Approach. E-kirja. Morgan Kaufmann Publishers.
- Dietrich, N. 2015. Snort 2.9.8.x on Ubuntu 12, 14, and 15. E-kirja sivutolla Snort.org. Viitattu 23.3.2016. <https://www.snort.org/documents>
- Dionicio, R. 2014. Installing Snorby for Snort. Julkaisu verkkosivustolla packet6.com. Julkaistu 13.8.2014. Viitattu 23.3.2016. <https://www.packet6.com/installing-snorby-for-snort/>
- Hayes, B., Scott, C., Wolfe, P. 2004. Snort For Dummies. E-kirja. John Wiley & Sons.
- Hedlug, B. 2011. On data center scale, OpenFlow, and SDN. Artikkelit bradhedlund.com-verkkosivustolla. Julkaistu 21.4.2011. Viitattu 20.1.2016. <http://bradhedlund.com/2011/04/21/data-center-scale-openflow-sdn/>
- Hoff, B. 2014. Five software-defined data center tips to consider before investing. Artikkelit SearchSDN –verkkosivulla. Julkaistu 5/2014. Viitattu 11.1.2016. <http://searchsdn.techtarget.com/tip/Five-software-defined-data-center-tips-to-consider-before-investing>.
- Hogg, S. 2014. SDN Security Attack Vectors and SDN Hardening. Artikkelit Networkworld-verkkosivulla. Julkaistu 28.10.2014. Viitattu 18.1.2016. <http://www.networkworld.com/article/2840273/sdn/sdn-security-attack-vectors-and-sdn-hardening.html>
- Inside SDN Architecture. n.d. Artikkelit SDX Central-verkkosivulla. Viitattu 12.1.2016. <https://www.sdxcentral.com/resources/sdn/inside-sdn-architecture/>
- Kuosmanen, P. 2015. DIGILE's Cyber Trust program waiting for take-off. Blogi-kirjoitus activityblog.fi-verkkosivulla. Julkaistu 16.02.2015. Viitattu 15.2.2016. <http://www.activityblog.fi/2015/02/digiles-cyber-trust-program-waiting-take/>
- L4-L7 Service Function Chaining Solution Architecture. 2015. Julkaisu opennetworking.org-sivustolla. Julkaistu 14.7.2015. Viitattu 8.4.2016. <https://www.opennetworking.org/sdn-resources/technical-library>.
- McGillicuddy, S. 2014. SDN security issues: How secure is the SDN stack? Artikkelit SearchSDN-verkkosivulla. Julkaistu 14.2.2014. Viitattu 18.1.2016. <http://searchsdn.techtarget.com/news/2240214438/SDN-security-issues-How-secure-is-the-SDN-stack>
- Mitchell, B. 2015. The Seven Layers of the OSI Model Illustrated . Julkaisu about.com-verkkosivustolla. Viitattu 14.4.2016. <http://compnetworking.about.com/od/osimodel/tp/The-Seven-Layers-of-the-OSI-Model-Illustrated.htm>

- ONF Review. n.d. ONF Overview. Organisaatio esittely [opennetworking.org/verkkosivulla](http://www.opennetworking.org/verkkosivulla). Viitattu 13.1.2016. <https://www.opennetworking.org/about/onf-overview>
- Openflow Switch Specificatio Version 1.5.1. 2015. Julkaisu [opennetworking.org/verkkosivustolla](http://www.opennetworking.org/verkkosivustolla). Julkaistu 26.3.2015. Viitattu 8.4.2016. <https://www.opennetworking.org/sdn-resources/technical-library>.
- OpenFlow. n.d. Artikkelit OpenFlow -verkkosivulla. Viitattu 12.1.2016. <http://archive.openflow.org/wp/learnmore/>
- Rouse, M. 2010. Virtual switch definition. Artikkelit SearchServerVirtualization-verkkosivulla. Julkaistu 7/2010. Viitattu 12.1.2016. <http://searchservvirtualization.techtarget.com/definition/virtual-switch>
- Rouse M. 2015. Software-defined networking (SDN) definition. Artikkelit SearchSDN-verkkosivulla. Viitattu 11.1.2016. <http://searchsdn.techtarget.com/definition/software-defined-networking-SDN>
- Ryu Project Team. 2014. RYU SDN Framework. E-kirja GitHub.io-verkkosivulla. Viitattu 6.4.2016. <http://osrg.github.io/ryu/resources.html#books>
- Sanders, C., Smith, J. 2014. Applied Network Security Monitoring: Collection, Detection, and Analysis. E-kirja. Syngress Publishing.
- Seagren, E. 2008. Secure Your Network for Free: Using Nmap, Wireshark, Snort, Nessus, and Mrtg. E-kirja. Syngress Publishing.
- Snort users manual. 2015. Snort-tiimin luoma käyttöopas. PDF-tiedosto [snort.org/verkkosivulla](http://www.snort.org/verkkosivulla). Viitattu 16.2.2016. <https://www.snort.org/documents/snort-users-manual>
- Software-Defined Networking (SDN) Definition. 2013. Artikkelit ONF-verkkosivulla. Julkaistu 1.5.2013. Viitattu 11.1.2016. <https://www.opennetworking.org/sdn-resources/sdn-definition>
- Sridhar, R. 2015. SDN Series Part Eight: Comparison Of Open Source SDN Controllers. Artikkelit TheNewStack-verkkosivulla. Viitattu 19.1.2016. <http://thenewstack.io/sdn-series-part-eight-comparison-of-open-source-sdn-controllers/>
- Sridhar, R. 2015. SDN Series Part Four: Ryu, a Rich-Featured Open Source SDN Controller Supported by NTT Labs. Artikkelit TheNewStack-verkkosivulla. Viitattu 14.1.2016. <http://thenewstack.io/sdn-series-part-iv-ryu-a-rich-featured-open-source-sdn-controller-supported-by-ntt-labs/>
- Stewart, M. 2014. Network Security, Firewalls and VPNs, Second Edition. Kappale 4: Network Security Threats and Issues. E-kirja. Jones and Bartless Learning.
- Thomas, J. n.d. Seven Layers of OSI Model and functions of seven layers of OSI model. Julkaisu [Omnisecu.com-verkkosivustolla](http://www.omnisecu.com/verkkosivustolla). Viitattu 30.1.2016. <http://www.omnisecu.com/tcpip/osi-model.php>

Ur Rehman, R. 2003. Intrusion Detection with SNORT: Advanced IDS Techniques Using SNORT, Apache, MySQL, PHP, and ACID. E-kirja. Prentice Hall.

What is OpenFlow? n.d. Artikkele SDX Central-verkkosivulla. Viitattu 12.1.2016. <https://www.sdxcentral.com/resources/sdn/what-is-openflow/>

What is Ryu Controller? n.d. Artikkele SDXcetrnal.com-verkkosivulla. Viitattu 19.1.2016. <https://www.sdxcentral.com/resources/sdn/sdn-controllers/open-source-sdn-controllers/what-is-ryu-controller/>.

What Is TCP Syn Flood Attack. 2015. Artikkele hackingaccount.com-verkkosivulla 4.5.2015. Viitattu 11.2.2016. <http://www.hackingaccount.com/what-is-tcp-syn-flood-attack/>.

What's a Software-Defined Data Center? n.d. Artikkele SDX Central-verkkosivulla. Viitattu 13.1.2016. <https://www.sdxcentral.com/resources/sdn/software-defined-data-center/>

Liitteet

Liite 1. Simple_switch_snort.py

```

# Copyright (C) 2013 Nippon Telegraph and Telephone Corporation.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from __future__ import print_function
import array

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ipv4
from ryu.lib.packet import icmp
from ryu.lib import snortlib

class SimpleSwitchSnort(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'snortlib': snortlib.SnortLib}

    def __init__(self, *args, **kwargs):
        super(SimpleSwitchSnort, self).__init__(*args, **kwargs)
        self.snort = kwargs['snortlib']
        self.snort_port = 3
        self.mac_to_port = {}

        socket_config = {'unixsock': True}

        self.snort.set_config(socket_config)

```

```

self.snort.start_socket_server()

def packet_print(self, pkt):
    pkt = packet.Packet(array.array('B', pkt))

    eth = pkt.get_protocol(ethernet.ethernet)
    _ipv4 = pkt.get_protocol(ipv4.ipv4)
    _icmp = pkt.get_protocol(icmp.icmp)

    if _icmp:
        self.logger.info("%r", _icmp)

    if _ipv4:
        self.logger.info("%r", _ipv4)

    if eth:
        self.logger.info("%r", eth)

    # for p in pkt.protocols:
    #     if hasattr(p, 'protocol_name') is False:
    #         break
    #     print('p: %s' % p.protocol_name)

@set_ev_cls(snortlib.EventAlert, MAIN_DISPATCHER)
def _dump_alert(self, ev):
    msg = ev.msg

    print('alertmsg: %s' % ".join(msg.alertmsg))

    self.packet_print(msg.pkt)

@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # install table-miss flow entry
    #
    # We specify NO BUFFER to max_len of the output action due to
    # OVS bug. At this moment, if we specify a lesser number, e.g.,
    # 128, OVS will send Packet-In with invalid buffer_id and
    # truncated packet data. In that case, we cannot output packets
    # correctly.
    match = parser.OFPMatch()
    actions = [parser.OFPACTIONOutput(ofproto.OFPP_CONTROLLER,
                                     ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)

```

```

def add_flow(self, datapath, priority, match, actions):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                        actions)]

    mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                            match=match, instructions=inst)
    datapath.send_msg(mod)

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]

    dst = eth.dst
    src = eth.src

    dpid = datapath.id
    self.mac_to_port.setdefault(dpid, {})

    # self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

    # learn a mac address to avoid FLOOD next time.
    self.mac_to_port[dpid][src] = in_port

    if dst in self.mac_to_port[dpid]:
        out_port = self.mac_to_port[dpid][dst]
    else:
        out_port = ofproto.OFPP_FLOOD

    actions = [parser.OFPActionOutput(out_port),
               parser.OFPActionOutput(self.snort_port)]

    # install a flow to avoid packet_in next time
    if out_port != ofproto.OFPP_FLOOD:
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
        self.add_flow(datapath, 1, match, actions)

    data = None
    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
        data = msg.data

```

```
out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,  
                           in_port=in_port, actions=actions, data=data)  
datapath.send_msg(out)
```


Liite 2. Snort.py

```

from __future__ import print_function

import ast
import array
import ryu.app.ofctl.api
import MySQLdb
import re
import ftplib
import urllib2

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.ofproto import ether
from ryu.ofproto import inet
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ipv4
from ryu.lib.packet import icmp
from ryu.lib import snortlib
from ryu.topology.api import get_switch
from time import sleep

class SimpleSwitchSnort(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'snortlib': snortlib.SnortLib}

    def __init__(self, *args, **kwargs):
        super(SimpleSwitchSnort, self).__init__(*args, **kwargs)
        self.snort = kwargs['snortlib']
        self.snort_port = 3
        self.mac_to_port = {}

        socket_config = {'unixsock': False}

        self.snort.set_config(socket_config)
        self.snort.start_socket_server()

    def packet_print(self, value, datapath, pkt):
        datapath = datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # Create dictionary of existing values

```

```

if value[1] in ('icmp'):
    ip_proto=1

if value[1] in ('tcp'):
    ip_proto=6

dict = {'eth_type': 2048};
dict['ip_proto'] = ip_proto;

if value[2] != ('any'):
    ipv4_src = value[2]
    dict['ipv4_src'] = ipv4_src;

if value[3] != ('any'):
    src_port = value[3]
    dict['tcp_src'] = int(src_port);

if value[5] != ('any'):
    ipv4_dst = value[5]
    dict['ipv4_dst'] = ipv4_dst;

if value[6] != ('any'):
    dst_port = value[6]
    dict['tcp_dst'] = int(dst_port);

# Use data of dictionary in OFPMatch parser
match = dict
match = parser.OFPMatch(**match)

self.send_flow_mod(datapath, match)

@set_ev_cls(snortlib.EventAlert, MAIN_DISPATCHER)
def _dump_alert(self, ev):
    msg = ev.msg

    # Get all dpid from switches
    sw_dpid = [s.dp.id for s in get_switch(self)]

    # Get datapath by dpid
    datapath = ryu.app.ofctl.api.get_datapath(self, sw_dpid[1])

    # Parse content
    pkt = msg.pkt
    pkt = packet.Packet(array.array('B', pkt))
    _ipv4 = pkt.get_protocol(ipv4.ipv4)
    alertmsg = ".join(msg.alertmsg)
    ip = _ipv4.src

```

```

id = _ipv4.identification

print('Received Alert from %s with alert message %s' % (ip, alertmsg))

self.get_snort_rule(datapath, id, pkt)

def get_snort_rule(self, datapath, id, pkt):
    db = MySQLdb.connect(host="192.168.1.79", # MySQL host
                        user="snorby",      # username
                        passwd="mysqlsnorby", # password
                        db="snorby")        # name of the data base

    cur = db.cursor()
    cur.execute("SELECT * FROM iphdr")
    cur.close()

    # Make a query to database
    cur = db.cursor()
    cur.execute("SELECT cid FROM iphdr WHERE ip_id = %s", (id))

    # get data from Snorby
    rows = cur.fetchall()
    for row in rows:
        cid = row
    cur.close()

    # Make another query
    cur = db.cursor()
    cur.execute("SELECT sig_name FROM events_with_join WHERE cid = %s", (cid))

    rows2 = cur.fetchall()
    for row2 in rows2:
        string = ".join(row2)

    cur.close()

    # Get the clean data from the string
    out = re.compile(':(.*?):', re.DOTALL | re.IGNORECASE).findall(string)
    if out :
        sid = ".join(out)

    # Open ftp connection
    ftp = ftplib.FTP('192.168.1.79', 'maria','akuanka89')

    # Get the rule file
    ftp.cwd("rules")
    gFile = open("ryu.rules", "wb")
    ftp.retrbinary('RETR ryu.rules', gFile.write)

```

```

gFile.close()
ftp.quit()

# Read the file contents
gFile = open("ryu.rules", "r")
buff = gFile.read()
gFile.close()

# Find the rule, that has correct sid
with open("ryu.rules") as f:
    for line in f:
        if sid in line:
            line2 = line

# Make a tuple than data can be compare to alert
value = tuple(line2.split(' '))

self.packet_print(value, datapath, pkt)

@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    match = parser.OFPMatch()
    actions = [parser.OFPACTIONOutput(ofproto.OFPP_CONTROLLER,
                                     ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)

def add_flow(self, datapath, priority, match, actions):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPIInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]

    mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                             match=match, instructions=inst)
    datapath.send_msg(mod)

# Create new flow to block packet next time
def send_flow_mod(self, datapath, match):
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

```

```

cookie = cookie_mask = 0
table_id = 0
idle_timeout = hard_timeout = 0
priority = 32768
buffer_id = ofp.OFP_NO_BUFFER
actions = [ofp_parser.OFPActionOutput(0)]
inst = [ofp_parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS,
                                         actions)]
req = ofp_parser.OFPFlowMod(datapath, cookie, cookie_mask,
                             table_id, ofp.OFPFC_ADD,
                             idle_timeout, hard_timeout,
                             priority, buffer_id,
                             ofp.OFPP_ANY, ofp.OFPG_ANY,
                             ofp.OFPFF_SEND_FLOW_REM,
                             match, inst)
datapath.send_msg(req)

```

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
```

```
def _packet_in_handler(self, ev):
```

```
    msg = ev.msg
```

```
    datapath = msg.datapath
```

```
    ofproto = datapath.ofproto
```

```
    parser = datapath.ofproto_parser
```

```
    in_port = msg.match['in_port']
```

```
    pkt = packet.Packet(msg.data)
```

```
    eth = pkt.get_protocols(ethernet.ethernet)[0]
```

```
    dst = eth.dst
```

```
    src = eth.src
```

```
    dpid = datapath.id
```

```
    self.mac_to_port.setdefault(dpid, {})
```

```
    # learn a mac address to avoid FLOOD next time.
```

```
    self.mac_to_port[dpid][src] = in_port
```

```
    if dst in self.mac_to_port[dpid]:
```

```
        out_port = self.mac_to_port[dpid][dst]
```

```
        else:
```

```
            out_port = ofproto.OFPP_FLOOD
```

```
    actions = [parser.OFPActionOutput(out_port),
```

```
               parser.OFPActionOutput(self.snort_port)]
```

```
    # install a flow to avoid packet_in next time
```

```
    if out_port != ofproto.OFPP_FLOOD:
```

```
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
```

```
self.add_flow(datapath, 1, match, actions)

data = None
if msg.buffer_id == ofproto.OFP_NO_BUFFER:
    data = msg.data

out = parser.OFPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                        in_port=in_port, actions=actions, data=data)
datapath.send_msg(out)
```

Liite 3. Snorby-tietokannan tietotauluja

sid	cid	ip_src	ip_dst	ip_ver	ip_hlen	ip_tos	ip_len	ip_id	ip_flags	ip_off	ip_ttl	ip_proto	ip_csum
1	157069	167772172	167772161	4	5	0	60	33840	0	0	64	6	57983
1	157070	167772172	167772161	4	5	0	60	33841	0	0	64	6	57982
1	157071	167772172	167772161	4	5	0	60	33842	0	0	64	6	57981
1	157072	167772172	167772161	4	5	0	60	33843	0	0	64	6	57980
1	157073	167772172	167772161	4	5	0	60	33844	0	0	64	6	57979

Kuvio 52. Iphdr-tietotaulu

```
mysql> select * from events_with_join limit 15;
```

sid	cid	signature	classification_id	users_count	user_id	notes_count	type	number_of_events	timestamp	id	ip_src	ip_dst	sig_priority	sig_name
1	157069	514	NULL	0	NULL	0	1	0	2016-04-06 01:43:56	157068	167772172	167772161	0	Snort Alert [1:1000006:1]
1	157070	514	NULL	0	NULL	0	1	0	2016-04-06 01:43:56	157069	167772172	167772161	0	Snort Alert [1:1000006:1]
1	157071	514	NULL	0	NULL	0	1	0	2016-04-06 01:43:57	157070	167772172	167772161	0	Snort Alert [1:1000006:1]
1	157072	514	NULL	0	NULL	0	1	0	2016-04-06 01:43:57	157071	167772172	167772161	0	Snort Alert [1:1000006:1]
1	157073	514	NULL	0	NULL	0	1	0	2016-04-06 01:43:57	157072	167772172	167772161	0	Snort Alert [1:1000006:1]
1	157074	514	NULL	0	NULL	0	1	0	2016-04-06 01:43:57	157073	167772172	167772161	0	Snort Alert [1:1000006:1]
1	157075	514	NULL	0	NULL	0	1	0	2016-04-06 01:43:58	157074	167772172	167772161	0	Snort Alert [1:1000006:1]
1	157076	514	NULL	0	NULL	0	1	0	2016-04-06 01:43:58	157075	167772172	167772161	0	Snort Alert [1:1000006:1]
1	157077	514	NULL	0	NULL	0	1	0	2016-04-06 01:43:58	157076	167772172	167772161	0	Snort Alert [1:1000006:1]

Kuvio 53. Events_with_join-tietotaulu

Liite 4. Kolmannen testin tulokset

Taulukko 10. Yhteenveto kolmannen testin tuloksista

Hyökkäys *	Host 1**	OpenFlow-drop ***
u10 000 (100 pkt/sek)		
550	139	451
640	137	547
652	276	502
964	496	512
538	131	434
u1000 (1000 pkt/sek)		
5030	1891	3970
5190	705	4546
4906	2065	3819
5050	990	4170
4519	623	3918
u100 (10 000 pkt/sek)		
16627	3291	13622
26452	10746	20546
31926	11958	20262
33580	12710	22063
28640	11848	17665
u10 (100 000 pkt/sek)		
82805	9876	74726
87107	20682	72014
80175	26607	54356
95424	35402	65855
138795	5577	123777
76324	19456	59084
74490	21689	55297
79300	30194	52725

* Mininet-verkkoon lähetettyjen TCP- pakettien kokonaismäärä

** Host 1-laitteelle saapuneiden TCP-pakettien kokonaismäärä

*** OpenFlow-vuosäännön pudottamien TCP-pakettien kokonaismäärä