T A R T U Ü L I K O O L
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Infotehnoloogia eriala

Alar Kvell

Suure jõudlusega sissetungi tuvastuse süsteemi lahendus S4A tarkvara jaoks

Magistritöö (30 EAP)

Juhendajad: Meelis Roos, MSc
Sven Heiberg, MSc

Autor: …… "….." mai 2012
Juhendaja: ……. "….." mai 2012

Lubada kaitsmisele
Professor : ……. "….." mai 2012

TARTU 2012

Sisukord

Sissejuhatus

Riigi Infosüsteemi Ameti (RIA) infoturbeintsidentide käsitlemise osakond tegutseb Eestis rahvusliku CERT (Computer Emergency Response Team) ülesannetes ning talitleb rahvusvahelise kontaktpunktina. CERT Eesti tuvastab, jälgib ja lahendab Eesti riigi arvutivõrkudes toimuvaid turvaintsidente, teavitab ohtudest ning korraldab ennetustegevusi.

S4A on võrguliikluse analüsaatori ja ründeprofiilide baasil loodud avatud lähtekoodiga tarkvara. Tarkvara võimaldab koguda keskselt statistikat võrguaadresside transleerimist (NAT) kasutavates sisevõrkudes esinevate turvarikete kohta, samas nende võrkude ehitust statistiku eest salajasse jättes. Selle tulemusena omandab CERT Eesti reaalajas pildi riigivõrgus aset leidvatest turvariketest. Kohaliku võrgu administraatorile annab S4A detailse pildi kohtvõrgus aset leidvatest turvariketest ning lihtsustab tunduvalt rikete lähtekoha leidmist.

S4A kasutab keskse komponendina tuvastusplatvormi, millel on operatsioonisüsteem OpenBSD ja sissetungi tuvastuse süsteem Snort. Sissetungi tuvastuse süsteemi (Intrusion Detection System ehk IDS) juures on oluliseks omaduseks selle jõudlus – et süsteem suudaks suure võrguliikluse mahu korral reaalajas analüüsida kogu võrguliikluse sisu või võimalikult suurt osa võrguliikluse sisust. Mida suurem osa võrguliikluse sisust analüüsimata jääb, seda rohkem ründeid jääb märkamata. Kui piisavalt suur osa võrguliikluse sisust jääb analüüsimata, siis on sissetungijal väga hea tõenäosus märkamatult tegutseda isegi tuntud ründemeetodit kasutades.

Põhiline valupunkt olemasoleva S4A lahenduse jõudluse juures on, et sissetungi tuvastuse tarkvara Snort on ühelõimeline ja seega ei suuda kasutada rohkem kui ühe protsessorituuma ressurssi. Riistvaralise jõudluse suurendamine on aga oluliselt odavam horisontaalselt skaleerides (suurendades protsessori tuumade arvu) kui vertikaalselt skaleerides (suurendades iga protsessorituuma jõudlust).

Käesoleva töö eesmärk on leida võimalusi S4A võrguliikluse sissetungi tuvastuse süsteemi

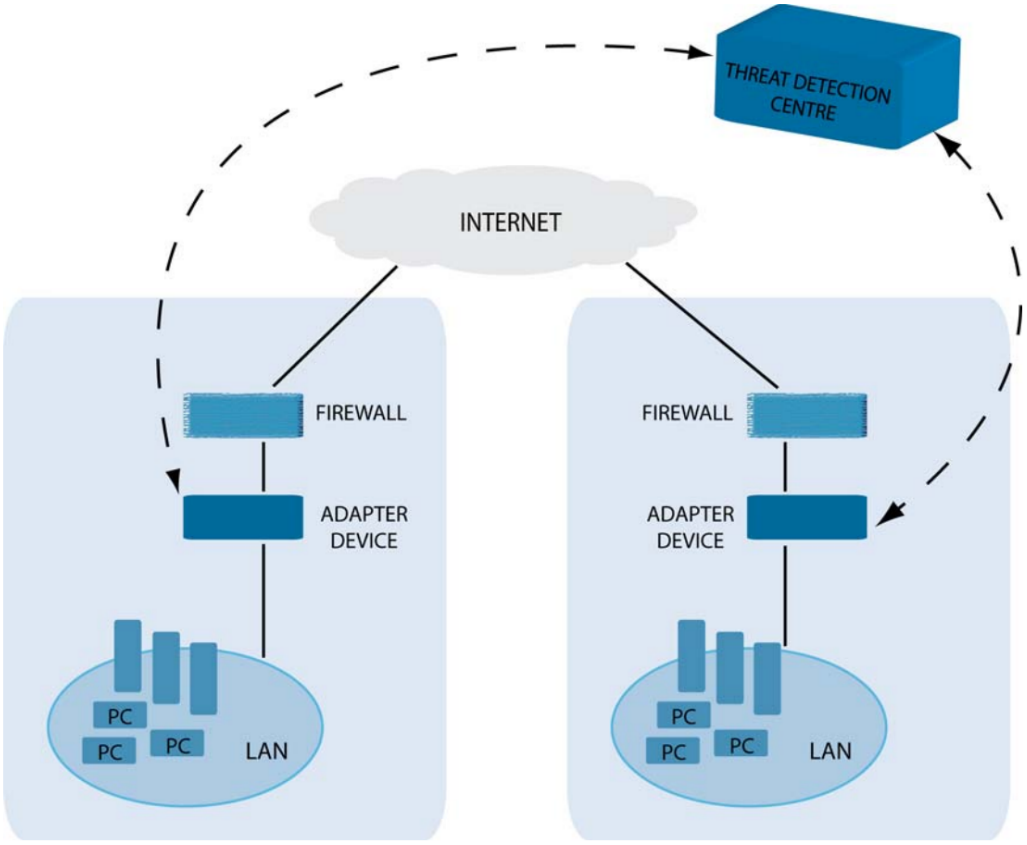
jõudluse suurendamiseks. Käesoleva töö käigus:

1. tutvutakse S4A ülesehitusega ja Snort tööpõhimõttega; viimase alusel analüüsitakse selle jõudlust mõjutavaid tegureid;
2. jõudlust mõjutavate tegurite alusel pakutakse välja erinevad võimalused, mida saab kaaluda S4A lahenduse jõudluse suurendamiseks;
3. mõõdetakse sissetungi tuvastuse süsteemi jõudluse muutumist võrguliikluse simulatsioonikeskkonnas pakutud lahenduste rakendamisel;
4. luuakse mõõtmistulemuste alusel võrdlus;
5. tehakse võrdluse alusel järeldused ja ettepanekud S4A lahenduse jaoks.

1 S4A arhitektuur

1.1 S4A arhitektuur ja tüüpiline juurutus

S4A (lühend sõnadest Snort for All) on turvarikete reaalajaline hajutatud jälgimissüsteem. See on loodud 2009. aastal ja seda arendab RIA tellimusel Cybernetica AS. S4A tarkvara on avatud lähtekoodiga ja tasuta kasutatav [1]. Dokumentatsioon avalikult kättesaadav ei ole.



Joonis 1. S4A jälgimissüsteemi arhitektuur [2].

S4A jälgimissüsteem koosneb ühest keskusest (joonisel 1 tähistatud *Threat Detection Centre*) ja ühest või mitmest tuvastajast (joonisel 1 tähistatud *Adapter Device*). Keskus on paigaldatud CERT Eestisse ning tuvastajad on paigaldatud erinevatesse riigiasutustustesse üle Eesti. Asutused on IT infrastruktuuri poolest väga erinevad.

Tuvastaja paikneb asutuse sisevõrgu äärel selliselt, et ta näeb mõlemas suunas võrguliiklust, mis asutuse sisevõrgu ja avaliku Interneti vahel toimub. Tuvastaja analüüsib võrguliiklust ja kogub selle kohta infot. Detailset infot saab vaadata kohaliku võrgu administraator ning üldistatud infot saadab tuvastaja perioodiliselt (iga 10 minuti tagant) keskusesse. Tuvastaja ja keskuse vahelised ühendused on krüpteeritud ja mõlemapoolselt autenditud.

Tuvastaja ülesandeks on analüüsida võrguliiklust ja tuvastada etteantud reeglite põhjal kahtlast liiklust. Kahtluse defineerib CERT Eesti, tüüpiliselt otsitakse kahjurvaraga seotud võrguliiklust ja protokollinõrkuseid ära kasutavat võrguliiklust. Tuvastaja ise otseselt ei kaitse rünnete eest, vaid teeb kogutud info kättesaadavaks inimestele. Tuvastaja toodetud infole reageerimise tulemusena on võimalik toimuvaid ründeid kiiremini avastada ja neile kiiremini reageerida ning tõhustada turvameetmeid toimumata rünnete ennetamiseks.

Tuvastaja koosneb ühest masinast, milles töötab ettenähtud tarkvara. Tuvastaja tarkvara koosneb:

1. OpenBSD operatsioonisüsteemist [3];
2. Snort sissetungi tuvastuse süsteemist [4];
3. S4A spetsiifilisest tugitarkvarast, mis analüüsib Snort'i väljundit, hoiab Snort'i töös, suhtleb keskusega ning sisaldab liideseid infole ligipääsuks ja süsteemi seadistamiseks.

Tuvastaja masinal on üks võrguliides halduseks ja üks kuni kaks võrguliidest analüüsitava võrguliikluse vastu võtmiseks. Asutuse võrgus on kommutaator (*switch*) seadistatud selliselt, et mõlemas suunas võrguliiklus, mis asutuse sisevõrgu ja avaliku Interneti vahel käib, kopeeritakse harundivõimelise (*mirror / spanning)* pordi kaudu tuvastaja masina vastavale võrguliidesele. Kui võrgus ei eksisteeri harundivõimelist kommutaatorit, siis saab kasutada eraldi riistvaralist harundit (*tap*). Snort loeb võrguliikluse hõlmelahenduse PCAP [5] kaudu kõik võrguliidesele saabuvad paketid, analüüsib neid ning kirjutab väljundina logi. Logi analüüsib S4A spetsiifiline tugitarkvara.

Tuvastaja masina tüüpiline riistvara on Dell PowerEdge R200 [6], selliseid tuvastajaid eksisteerib Eestis suurusjärgus 100. Nimetatud masinas on 2 CPU tuuma ja selles töötab OpenBSD mitme protsessori toega (MP) tuum. Võrguliidesed on 1 Gbit/s ribalaiusega.

Snort tuvastusreegleid haldab CERT Eesti ning tüüpiliselt on kasutusel suurusjärgus 1500 reeglit.

Käesolevas töös vaadeldakse lähemalt ainult tuvastaja koosseisus olevat sissetungi tuvastuse süsteemi (Snort) ja sellega seotud operatsioonisüsteemi ja riistvara küsimusi. Muid S4A süsteemi osasid käesoleva töö raames ei vaadelda.

1.2 Snort tööpõhimõte

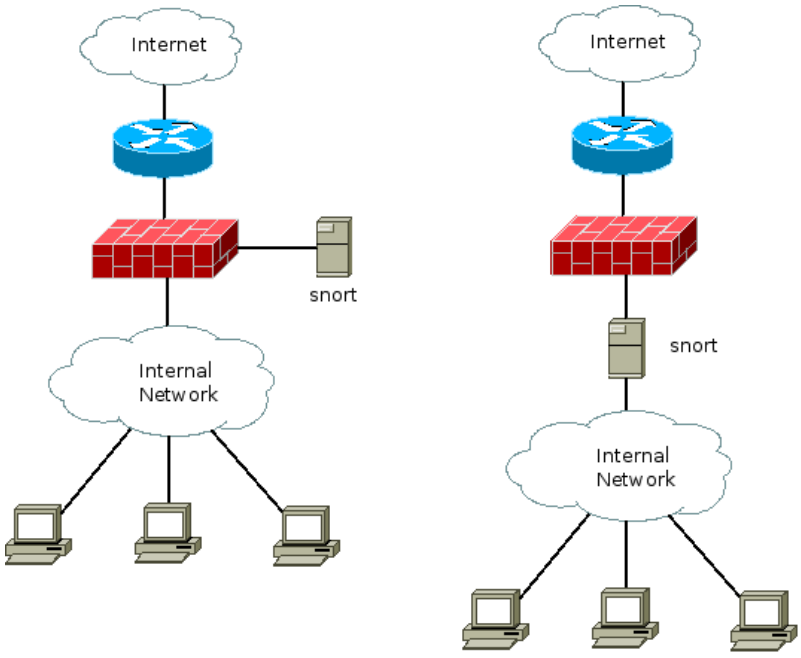
Selleks, et välja pakkuda võimalusi jõudluse suurendamiseks, tutvume esmalt Snort'i tööpõhimõttega ja analüüsime, millest Snort'i jõudlus sõltub.

Snort on avatud lähtekoodiga ja tasuta kasutatav sissetungi tuvastuse süsteem. Snort on loodud 1998. aastal ning seda arendab USA firma Sourcefire.

Snort'il on kaks olulisemat töörežiimi:

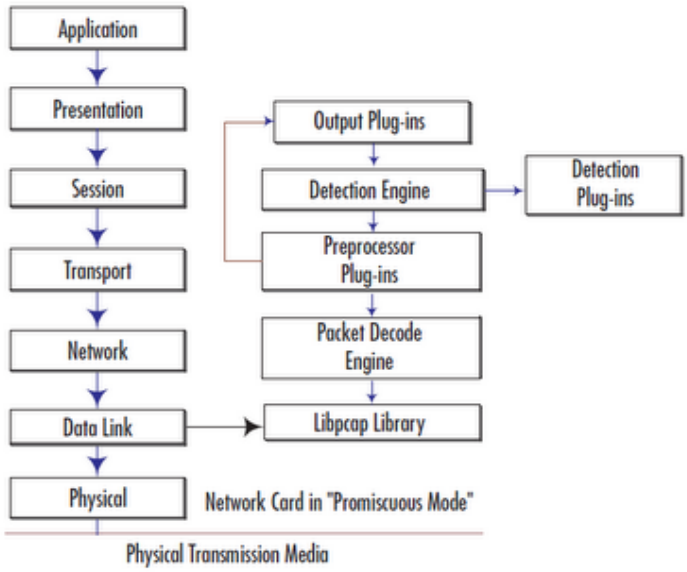
1. passiivne (*passive*) – ainult loeb võrgupakette.
2. sekkuv (*inline*) – lisaks võrguliikluse lugemisele võib seda ka blokeerida.

Snort'i saab võrgutopoloogiasse paigutada kahel viisil, mis on toodud joonisel 2. Mõlema paigutuse puhul saab kasutada kumbagi töörežiimi.



Joonis 2. Snort'i paiknemise valikud võrgutopoloogias [7].

S4A süsteemis kasutatakse Snort passiivset režiimi ja esimest paiknemise viisi, kus võrguliiklus ei läbi Snort masinat, vaid peegeldatakse sellele.



Joonis 3. Snort arhitektuur [8].

Snort loeb võrguliidesest pakette OSI mudeli 2. kihi (kanalikihi) tasemel (tüüpiliselt Ethernet kaadrid (*frame*)), kasutades selleks PCAP või analoogset mehhanismi.
Snort rekonstrueerib Ethernet kaadritest esmalt 3. kihi (võrgukihi) andmed – dekodeerib IP ja defragmenteerib protokolli paketid. Seejärel rekonstrueerib Snort sellest omakorda 4. kihi (võrgukihi) andmed – rekonstrueerib TCP ja UDP vood. UDP vood pole formaalselt defineeritud nagu TCP vood on, aga praktikas kasutatakse sama lähte-IP-aadressi ja pordi ja siht-IP-aadressi ja pordiga UDP pakette sama UDP voona.

Lisaks sisaldab Snort ka kõige levinumate kõrgema taseme protokollide (nt. HTTP, SMTP jm.) töötlemise võimekust. Erinevaid protokolle töötlevaid osasid nimetatakse Snort'is eel-töötlejateks (*preprocessor plug-in*) (vt. joonis 3). Eeltöötlejatel on 3 omadust:

1. Eeltöötlejad rekonstrueerivad infot, mille teevad kättesaadavaks kõrgema taseme eeltöötlejatele ja/või reeglite tuvastuse mootorile. Näiteks HTTP eeltöötleja pakib lahti HTTP sisus leiduda võiva pakitud voo ning normaliseerib JavaScript'i jpm. Eeltöötlejad võivad mingit osa liiklust ka mitte töödelda ja seda ka mitte anda edasi kõrgema taseme eeltöötlejatele või reeglitetuvastusmootorile;
2. Eeltöötlejad võivad ka ise toota mõningaid hoiatusi;
3. Eelmiste punktide käitumist saab mõjutada seadistustega.

Reeglite tuvastuse mootori ülesandeks on reeglite kõrvutamine eeltöötlejate poolt kättesaadavaks tehtud infoga, et leida vasteid. Reegel on kindla süntaksiga kirjutatud muster. Reegel koosneb väljadest (*rule options*), millest mõned on üldised ja mõned on eeltöötlejate spetsiifilised. Kõik reeglid laetakse muutmällu programmi käivitamisel. Tüüpiliselt on reegleid suurusjärgus 1000 – 10 000.

Kui töödeldav info vastab mõnele reeglile, siis genereeritakse selle kohta hoiatus. Hoiatus koosneb järgnevast infost:

- hoiatuse genereerimise aeg;
- reegel, millele vastavus leiti;
- võrgupaketi info – minimaalselt võrguprotokoll, lähte-IP-aadress ja port, siht-IP-aadress ja port; soovi korral ka rohkem infot.

Tegemist on reaalajas töötava süsteemiga. Kui võrgupakette ei jõuta nii kiiresti läbi vaadata kui neid peale tuleb, siis need jäetakse vahele (*drop*). Eeltöötlejate seadetest sõltub, kui palju ja millist liiklust reaalselt läbi vaadatakse. Alamhulga töötlemise eesmärgiks on kompromiss – kuna töötlemine on CPU ressursi nõudlik töö ning CPU ressurss on väärtuslik, siis otstarbekas on keskenduda ainult kõige olulisema info töötlemisele.

Eelneva info põhjal saame järeldada, et Snort jõudlus sõltub järgnevatest teguritest:

1. Sisendandmete ehk võrguliikluse ehk ribalaius;
2. Tarkvara implementatsioon:
   2.1. Snort kood;
   2.2. Operatsioonisüsteemi ja võrgupakettide hõlmelahenduse kood.
3. Riistvara jõudlus (võrgukaart, CPU, muutmälu ja nende omavahelised ühendused);
4. Reeglite arv;
5. Eeltöötlejate seadistused:
   5.1. Seadistused, mis muudavad töödeldava info hulka;
   5.2. Seadistused, mis ei muuda töödeldava info hulka (muudavad nt. töötlemisalgoritme);

Snort'i puhul toome välja, et tegemist on ühelõimelise protsessiga, mistõttu üks Snort protsess kasutab korraga maksimaalselt ühe CPU tuuma ressurssi.

1.3 Suricata tööpõhimõte

Vaatleme alternatiivset sissetungi tuvastuse süsteemi Suricata, mida töös hiljem kaalume Snort asendajana. Siin vaatleme, mille poolest see on Snort'iga sarnane ja mille poolest erineb.

Suricata on avatud lähtekoodiga ja tasuta kasutatav sissetungi tuvastuse süsteem. Suricata on loodud 2009. aastal ning seda arendab USA organisatsioon Open Information Security Foundation.

Suricata'l on olemas samad töörežiimid, mis Snort'il ning seda saab paigutada võrgu-topoloogiasse samadel viisidel. Suricata loeb võrguliiklust samal põhimõttel ning toetab paljusid samu hõlmelahendusi, mis Snort.

Suricata kasutab samu tuvastusreegleid ja toodab samas formaadis väljundit, mis Snort. Kui olla Snort'iga tuttav, siis saab Suricata't piisavalt väikese vaevaga võtta kasutusele Snort'i asemele või kõrvale.

Suricata ülesehituse üldpõhimõte on sama, mis Snort'il – rekonstrueeritakse võrguliikluse pakettidest kõrgema taseme info, mida tehakse kättesaadavaks reeglite tuvastamise mootorile. Suricata realisatsioon erineb Snort'ist märgatavalt, mistõttu Suricata võib anda erinevaid tulemusi (näiteks teeb teistsuguseid kompromisse võrguliikluse pakettide töötlemata jätmisel vm.).

Kuna Suricata ülesehituse üldpõhimõte on sama, siis mõjutavad selle jõudlust samad tegurid, mis Snort'i.

2 Võimalused sissetungi tuvastuse süsteemi jõudluse suurendamiseks

Sissetungi tuvastuse süsteemi juures on oluliseks omaduseks selle jõudlus – et süsteem suudaks suure võrguliikluse mahu korral reaalajas analüüsida kogu võrguliikluse sisu või võimalikult suurt osa võrguliikluse sisust. Mida suurem osa võrguliikluse sisust analüüsimata jääb, seda rohkem ründeid jääb märkamata. Kui piisavalt suur osa võrguliikluse sisust jääb analüüsimata, siis on sissetungijal väga hea tõenäosus märkamatult tegutseda isegi tuntud ründemeetodit kasutades.

Käesoleva töö eesmärgiks on suurendada S4A sissetungi tuvastuse tarkvara jõudlust. Jõudlus on defineeritud kui analüüsitava võrguliikluse hulk ühes ajaühikus. Peame silmas võrgupakettide arvu, mida tarkvara jõuab hõlmelahenduse kaudu sisse lugeda. Kui tarkvara kõrgemates kihtides jätab mingeid andmeid analüüsimata, siis seda ei arvestata analüüsitud võrgupakettide arvust maha, vaid see liiklus jääb analüüsitud võrgupakettide arvu sisse.

Järgnevalt vaatleme iga peatükis 1.2 kirjeldatud teguri muutmist iseseisvalt.

Tegur 1. Organisatsiooni summaarse võrguliikluse hulga vähendamine ei ole organisatsioonilistel põhjustel võimalik. Ühe tuvastaja töödeldava võrguliikluse hulka on võimalik vähendada, kui paigaldada mitu tuvastajat, igaüks eraldi alamvõrku. Siis rakendub igale tuvastajale erinev alamhulk summaarsest võrguliikluse hulgast.

Tegur 2. Muudame tarkvara implementatsiooni. Pakume välja järgmised suunad:

1. Kasutada Snort'i Linux tuumaga [9] operatsioonisüsteemil. OpenBSD operatsioonisüsteemi põhirõhk on turvalisusel, mitte jõudlusel.
2. Panna Snort kõikide CPU tuumade peal tööle. Selleks on vaja Linux'il kasutada võrgupakettide hõlmelahendust PF_RING [10]. Viimane võimaldab omavahel seotud võrguliikluse pakette grupeerida voogudesse ning mitmel programmil lugeda pakette selliselt, et igaüks saab erinevad vood. Panna iga CPU tuuma kohta

üks Snort protsessi lugema PF_RING'ist, seega saab kõik protsessori tuumad ära kasutada.

3. Kasutada Snort alternatiivina sissetungi tuvastuse süsteemi Suricata [11]. Suricata on mitmelõimeline, seega saab ära kasutada kõiki tuumasid.
4. Kasutada Suricata puhul graafikaprotsessori (GPU) tuge. Suricata toetab CUDA võimeliste [12] graafikakaartide korral teatud hulga töö viimist CPU pealt GPU peale.

Nimetatud suunade põhjal pakume välja järgnevad konkreetsed tarkvara muutmise kombinatsioonid, mida käesolevas töös nimetame implementatsioonideks:

1. OpenBSD'l jookseb ühe protsessori toega tuum ja töötab Snort, mis kasutab võrgupakettide hõlmelahendust PCAP;
2. OpenBSD'l jookseb ühe protsessori toega tuum ja töötab Suricata, mis kasutab võrgupakettide hõlmelahendust PCAP;
3. OpenBSD'l jookseb mitme protsessori toega tuum ja töötab Suricata, mis kasutab võrgupakettide hõlmelahendust PCAP;
4. Linux'il on teine protsessori tuum välja lülitatud ja töötab Snort, mis kasutab võrgupakettide hõlmelahendust AF_PACKET;
5. Linux'il töötab kaks Snort protsessi, mis kasutavad võrgupakettide hõlmelahendust PF_RING;
6. Linux'il on teine protsessori tuum välja lülitatud ja töötab Suricata, mis kasutab võrgupakettide hõlmelahendust AF_PACKET;
7. Linux'il töötab Suricata, mis kasutab võrgupakettide hõlmelahendust AF_PACKET;
8. Linux'il töötab Suricata, mis kasutab võrgupakettide hõlmelahendust AF_PACKET ning GPU tuge.

Ülevaatlikuse eesmärgil esitame eelnevalt loetletud implementatsioonid ka tabelis 1 ja tabelis 2.

Tabel 1. Jõudluse mõõtmise Snort implementatsioonid.

	OpenBSD	**Linux**
1 CPU	X	X
2 CPU		X
GPU		

Tabel 2. Jõudluse mõõtmise Suricata implementatsioonid.

	OpenBSD	**Linux**
1 CPU	X	X
2 CPU	X	X
GPU		X

Tabelites tühjade lahtritega tähistatud implementatsioone ei ole võimalik konstrueerida järgnevatel põhjustel:

- OpenBSD'l ei ole võrgupakettide hõlmelahendust (PF_RING või analoogi), mis võimaldaks üksteisega seotud võrgupakette jagada voogudesse ja erinevaid vooge tarbida erinevatel programmidel;
- OpenBSD'l ei ole Nvidia CUDA tuge, seega ei saa Suricata GPU tuvastajat kasutada;
- Snort'il ei ole graafikaprotsessori tuge. Varasemate tööde käigus on laiendatud Snort tarkvara nimetatud toega [13] [14], kuid tulemuseks saadud tarkvara ei ole avalikult kättesaadav.

Välja pakutud implementatsioonide omavaheliseks võrdlemiseks jätame ülejäänud tegurid 3-5 (riistvara, reeglite arv, eeltöötlejate seadistused) samaks ning iga implementatsiooni korral tahame leida, kuidas võrguliikluse ribalaiusest sõltuvad järgnevad omadused:

1. analüüsitud võrgupakettide arv võrreldes kogu võrgupakettide arvuga;
2. leitud hoiatuste arv võrreldes antud võrguliiklusest antud programmiga maksimaalselt leitavate hoiatuste arvuga;
3. CPU / GPU koormus;
4. muutmälu hõivatus;
5. kõvaketta I/O koormus.

Püstitame hüpoteesi, et kõikide implementatsioonide kohta kehtivad järgnevad omadused:

1. Võrguliikluse ribalaiuse kasvades kasvab CPU koormus.
2. Kui CPU koormus ei küündi 100%-ni, siis on analüüsitud pakettide arv 100% ja hoiatuste arv 100%. Kui CPU koormus on 100% lähedal, siis võrguliikluse ribalaiuse kasvades väheneb analüüsitud pakettide arv ja väheneb hoiatuste arv.
3. Muutmälu hõivatus ei sõltu võrguliikluse ribalaiusest.

Tegur 3. Riistvara jõudluse suurenedes suureneks tarkvara jõudlus. Riistvara jõudluse suurendamine toob kaasa olulisi kulusid. Käesolevas töös seda täpsemalt ei käsitleta.

Tegur 4. Reeglite arvu vähendamisel suureneks tarkvara jõudlus. Reeglite arvu vähendamine ei ole otstarbekas, kuna siis väheneb tuvastatavate sissetungide arv.

Tegur 5. Tuleb leida, kas kummalgi tarkvaral on selliseid seadistusi, mis oluliselt mõjutavad jõudlust, aga ei muuda töödeldava info hulka. Varasemalt on leitud [15], et Suricata puhul eksisteerib kaks sellist seadistust, mis märgatavalt mõjutavad jõudlust: *mpm-algo* (*Multi Pattern Algorithm*) ja *detect-engine profile*. Testimise käigus selgus, et mõlema seadistuse selliste väärtuste korral, mis peaksid jõudlust parandama, tahtis Suricata hõivata testmasinas rohkem muutmälu kui masinas eksisteeris. Seetõttu jätame käesoleva töö raames seadistuste varieerimise kõrvale ja kasutame kummagi tarkvara puhul vaikeseadeid (välja arvatud seal, kus on teisiti kirjeldatud).

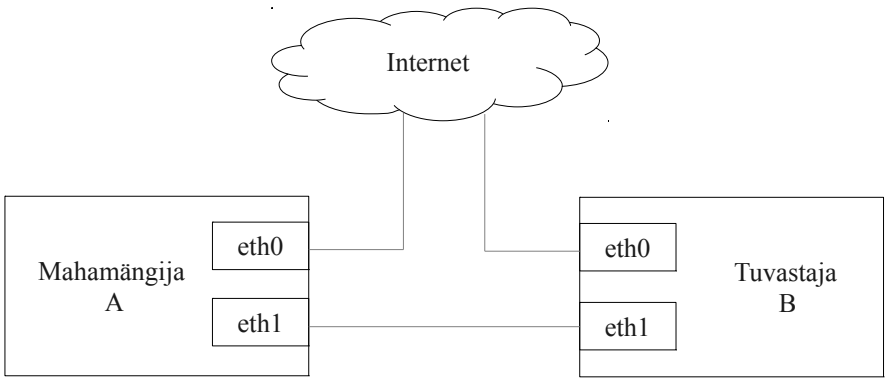
3 Mõõtmised

3.1 Mõõtmiste sooritamise keskkond ja lähteandmed

Varasemalt on Snort ja Suricata võrdlemiseks sooritatud mõõtmistel kasutatud sisendina nii reaalsest võrgust peegeldatud või salvestatud võrguliiklust muutmata kujul [16] kui ka sellist võrguliiklust, millele on tehislikult lisatud liiklust, mille tuvastust soovitakse kontrollida [17] [18]. Käesolevas töös kasutame sisendina reaalsest võrgust salvestatud liiklust muutmata kujul, kuna autoril oli piisavalt lihtne omandada sellist liiklust, mille pealt on võimalik piisavalt suur hulk hoiatusi tuvastada.

Eelnevalt on leitud [19], et kui kõrvale jätta libpcap teegi modifitseerimine, siis kõige suurema efektiivsusega mehhanism Linux'il PCAP failist pakettide võrku maha mängimiseks on tcpreplay [20], mis kasutab PF_PACKET toore sokli *send* mehhanismi.

Erinevate implementatsioonide kohta jõudluse mõõtmised sooritatame keskkonnas, mis koosneb ühest võrguliikluse mahamängija masinast A ja ühest tuvastaja masinast B (joonis 4).



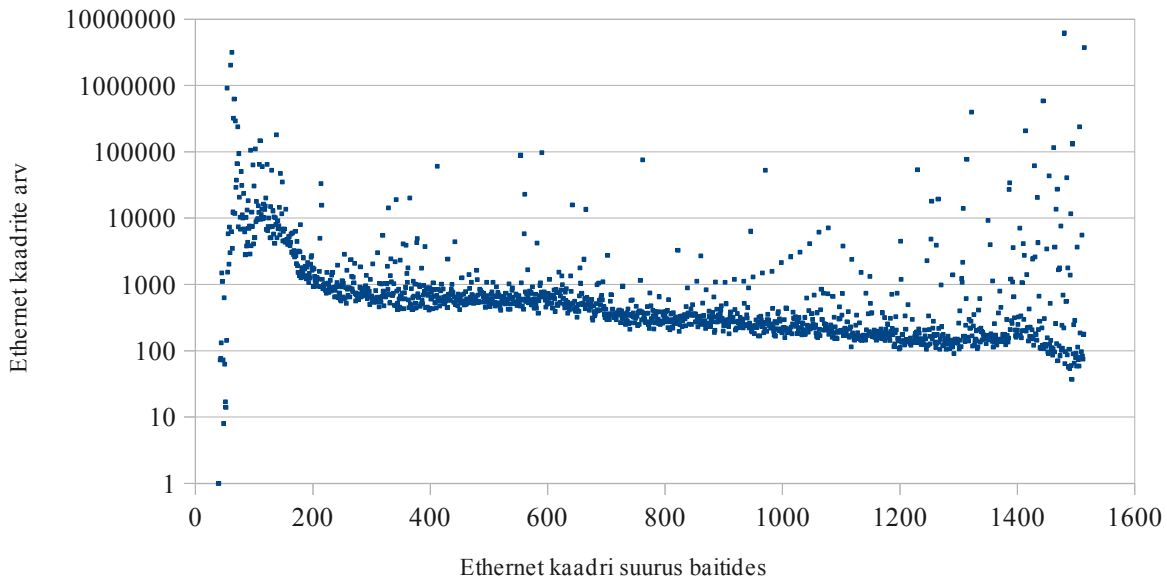
Joonis 4. Jõudluse mõõtmise keskkonna ülesehitus.

Kummalgi masinal on üks avalik ja üks sisemine Etherneti võrguliides. Avalike võrguliideste kaudu toimub mõõtmiste juhtimine. Kahe masina sisemised võrguliidesed on omavahel CAT5E võrgukaabliga otse kokku ühendatud. Mahamängija masin teostab

võrguliikluse mahamängimise sisemise võrguliidese peale, kasutades programmi tcpreplay. Tuvastaja masinas loeb IDS sisemise võrguliidese pealt liiklust. Täpsemad parameetrid tuvastaja keskkonna reprodutseerimiseks on toodud lisas 1.

Mõõteseeriate võrguliikluse ribalaiuse valiku jaoks on oluline teada, mis on maha-mängimise maksimumribalaius. Selleks teostame 5 mõõtmist, andes tcpreplay programmile ette ribalaiuse, millest alates mahamängimise ribalaius enam ei suurene. Tuvastaja masinas (Linux) mõõtes saime vastu võetava võrguliikluse ribalaiuse aritmeetiliseks keskmiseks 449,7 Mbit/s (standardhälve 14,5 Mbit/s).

Võrguliikluse sisendina kasutame salvestatud liiklust ühest reaalsest võrgust, kus on palju tudengitest kasutajaid. Võrguliiklus on salvestatud PCAP formaadis faili kujul, mis sisaldab Etherneti kaadreid. Käesoleva töö jaoks valime sealt esimesed 23 miljonit Ethernet kaadrit, mille sisu kogumahuks on 19,38 GB. Valik on tehtud arvestusega, et maksimumribalaiusel mahamängides tuleks mõõtmise kestvuseks vähemalt 5 minutit.



Joonis 5. Mõõtmisteks valitud võrguliikluse Etherneti kaadrite suuruste jaotus.

Mõõtmiseks valitud võrguliikluse hulgas on maksimaalseks Ethernet kaadri suuruseks 1514 baiti. Jooniselt 5 on näha, et valitud võrguliiklus sisaldab selles vahemikus igas

suuruses Ethernet kaadreid, mis täidab reaalse juhusliku võrguliikluse eesmärki hästi.

Reeglite sisendiks on valitud tasuta kättesaadavate reeglite hulgast [21] maksimaalne hulk selliseid, mis on mõlema programmi puhul toetatud. Selle tulemusena saame kokku 5376 reeglit. Testimise käigus selgus, et 3 reeglit tuleb erinevatel põhjustel eemaldada, seega saame kokku 5373 reeglit. Reeglite arvu me ei vähenda, kuna soovime mahamängija maksimaalse võrguliikluse ribalaiuse juures jõuda tuvastajas kõikide implementatsioonide puhul maksimaalse CPU koormuseni.

3.2 Snort ja Suricata täpsuse mõõtmised

Selleks, et saaks võrrelda Snort ja Suricata jõudlust, on vaja kindlaks teha, kas need sama sisendinfo korral toodavad sama väljundit, tingimusel et kogu võrguliiklus jõutakse ära töödelda.

Kasutame kummagi programmi puhul vaikeseadistusi, mis programmiga kaasa tulid. Lisaks sellele muudame reeglitega seotud seadistused programmide vahel identseks. Võtame samad sisendandmed (23 miljonit Ethernet kaadrit) ja samad reeglid (5373 reeglit), mida jõudluse mõõtmiste juures kasutame, ning söödame kummalegi programmile ette ühenduseta (*offline*) režiimis. Ühenduseta režiimi puhul pole reaalajalisus oluline – võrgupaketid loetakse PCAP failist ning alati raporteeritakse, et kõik paketid töödeldi ära. Seega võrdleme ainult hoiatuste arvu kahe programmi vahel.

Mõõtmiste käigus selgus, et Snort jätab mingi osa voogusid analüüsimata, kuna teatud mälupiir saab täis (Snort logis tuleb selle kohta veateade „*S5: Pruned X sessions from cache for memcap.*“) ning see mälupiir sõltub seadistusest *stream5_global memcap*. Suricata puhul selgus analoogne käitumine, et mingi osa voogusid jäetakse analüüsimata, kuna teatud mälupiir saab täis (Suricata logis tuleb selle kohta veateade „*Warning, engine running with FLOW_EMERGENCY bit set*“) ning see mälupiir sõltub seadistusest *flow memcap*.

Tabel 3. Snort ja Suricata hoiatuste ja veateadete arv sõltuvalt mälupiiri seadistusest.

Mälupiir	**Hoiatuste arv**		**Hoiatuste vahe**	**Identsete hoiatuste arv**	**Veateadete arv**	
	Snort	**Suricata**			**Snort**	**Suricata**
32 MiB	63606 (108)	64572 (97)	966	59283 (83)	14915	4
64 MiB	64293 (109)	64680 (94)	380	59914 (84)	0	0
128 MiB	64418 (109)	64684 (94)	266	59989 (85)	0	0
256 MiB	64419 (109)	64693 (97)	274	59991 (85)	0	0

Tabelis 3 on toodud Snort ja Suricata hoiatuste ja veateadete arv sõltuvalt mälupiiri seadistusest. Sulgudes on toodud ainulaadsete hoiatuste arv.

Lähtuvalt leitud tulemustest valime järgnevas peatükis jõudlusmõõtmiste jaoks nii Snort kui Suricata mälupiiri väärtuseks 64 MiB. Valik on tehtud arvestades, et kõrgema väärtuse puhul hoiatuste arvu vahe ei vähene enam oluliselt ning veateadete arv alates valitud väärtusest ja kõrgemal on null.

Identsete hoiatuste arv on suur (93% mälupiiri 64-256 MiB puhul), seega järgnevates peatükkides tehtavate mõõtmiste jaoks on Snort ja Suricata omavaheline täpsus piisavalt hästi võrreldav.

3.3 Jõudluse mõõtmised

Jõudluse mõõtmiseks teostame iga implementatsiooni kohta mitu mõõtmiste seeriat, iga seeria erineva võrguliikluse ribalaiusega. Kõikide mõõteseeriate korral anname sisendiks identsed andmed, mida kirjeldati peatükis 3.1. Igas mõõtmiste seerias sooritame 5 identset mõõtmist.

Iga mõõtmise puhul jälgime tuvastaja masinas 1 sekundilise intervalliga järgmiseid näitajaid:

- võrguliikluse ribalaius;
- CPU koormus;
- muutmälu hõivatus;
- kõvaketta I/O koormus.

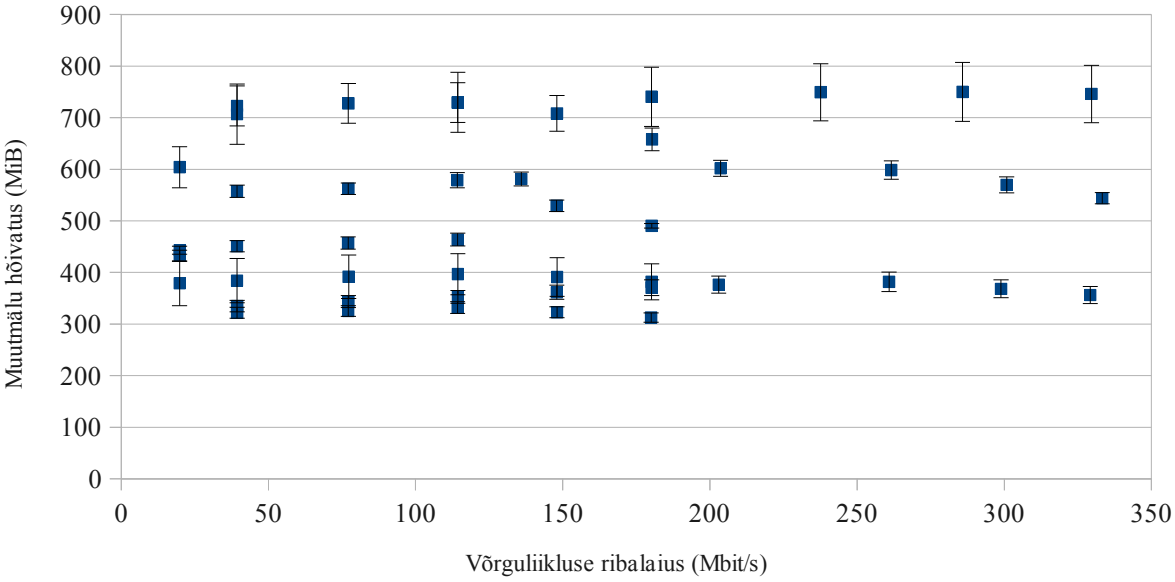
Iga nimetatud näitaja kohta arvutame väärtuste aritmeetilise keskmise ja standardhälbe, et hinnata mõõtmiste varieeruvust. Lisaks leiame iga mõõtmise puhul järgmised näitajad tuvastaja masinas:

- kõikide pakettide arv Snort / Suricata programmis;
- analüüsitud pakettide arv;
- tuvastatud hoiatuste arv.

Mõõtmiste tulemusena selgus, et kõikide pakettide arv Snort / Suricata programmis vastas iga mõõtmise puhul saadetud pakettide arvule tcpreplay programmi poolt, seega riistvara ega operatsioonisüsteemi poolt ühtegi paketti kaduma ei läinud ning pakettide analüüsimata jätmine toimus puhtalt Snort / Suricata programmi siseselt.

Mõõtmiste tulemusena selgus, et kõvakettalt lugemise kiirus oli 0,0 Mbit/s (standardhälve 0,0 Mbit/s) ja kõvakettale kirjutamise kiirus oli 0,2 Mbit/s (standardhälve 1,3 Mbit/s). Kuna mõõtmiste käigus avaldunud kõvaketta I/O koormus on kõvaketta maksimaalse saavutatava I/O koormusega (suurusjärgus 400-500 Mbit/s) võrreldes väga väike, siis järeldame, et see ei mõjuta jõudlust ja seda täpsemalt ei analüüsi.

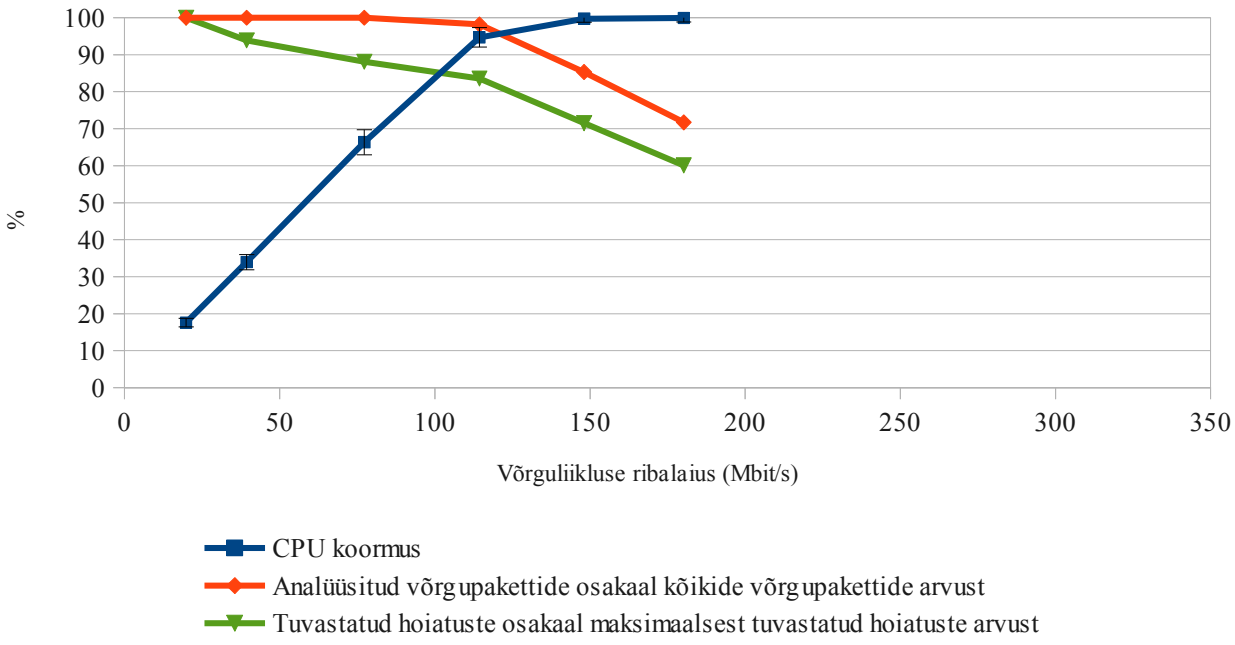
Mõõtmiste tulemusena selgus, et muutmälu hõivatus ei sõltu võrguliikluse ribalaiusest (joonis 6).



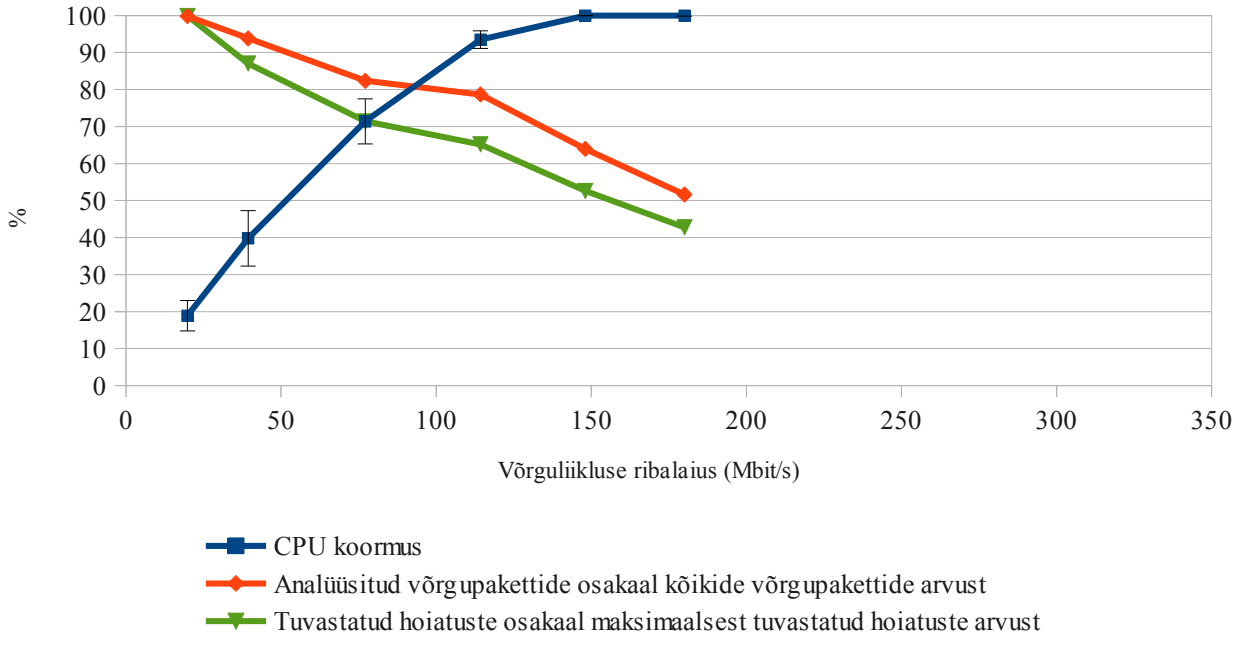
Joonis 6. Muutmälu hõivatuse sõltuvus võrguliikluse ribalaiusest.

Mõõtmiste käigus selgus, et maksimaalne tuvastatud hoiatuste arv erineb eelnevas peatükis täpsuse eesmärgil tehtud mõõtmiste käigus leitud hoiatuste arvust. Snort puhul oli maksimaalne tuvastatud hoiatuste arv 1,3 korda suurem, Suricata puhul 1,4 korda suurem. Nimetatud erinevuse lähem analüüs jäi praeguse töö eesmärkidest välja, kuid väärib kindlasti lähemat uurimist edasises töös. Omavahel erinesid Snort ja Suricata maksimaalne tuvastatud hoiatuste arv 1,1 korda, mis on piisavalt väike selleks, et jätkata sama järeldusega, mis peatüki 3.1 lõpus tehti – järgnevate mõõtmiste jaoks on Snort ja Suricata omavaheline täpsus piisavalt hästi võrreldav.

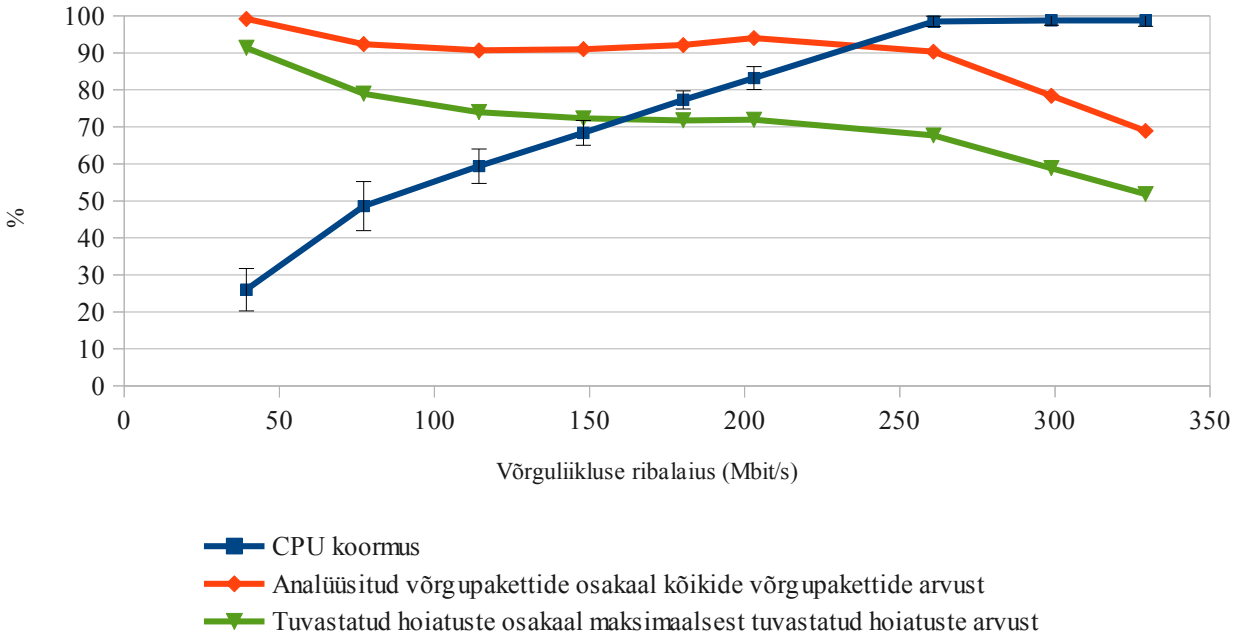
Järgnevalt on joonistel 7, 8, 9, 10, 11, 12, 13 toodud implementatsioonide 1-7 jõudluse mõõtmistulemused. Implementatsioon 8 (GPU toega) mõõtmisi ei saanud teostada, kuna selgus, et mõõtmiseks valitud reeglite arvu juures hõivas Suricata rohkem muutmälu kui masinas füüsiliselt kokku oli (4 GiB).



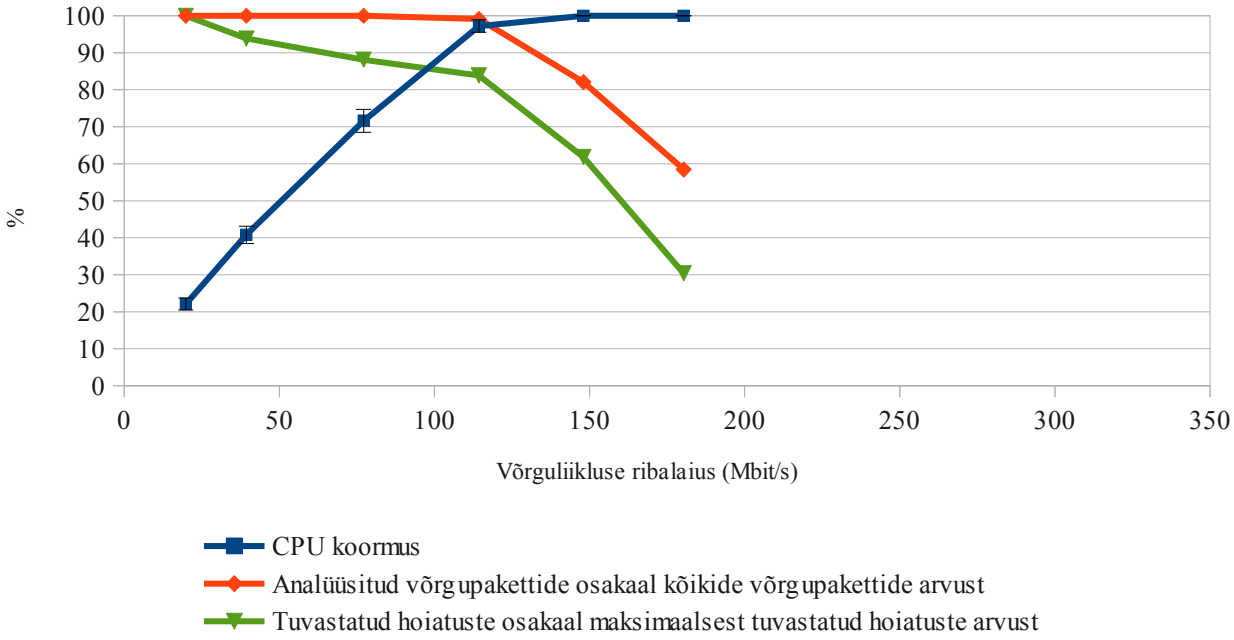
Joonis 7. Jõudluse mõõtmise implementatsioon 1: Snort, PCAP, 1 CPU, OpenBSD.



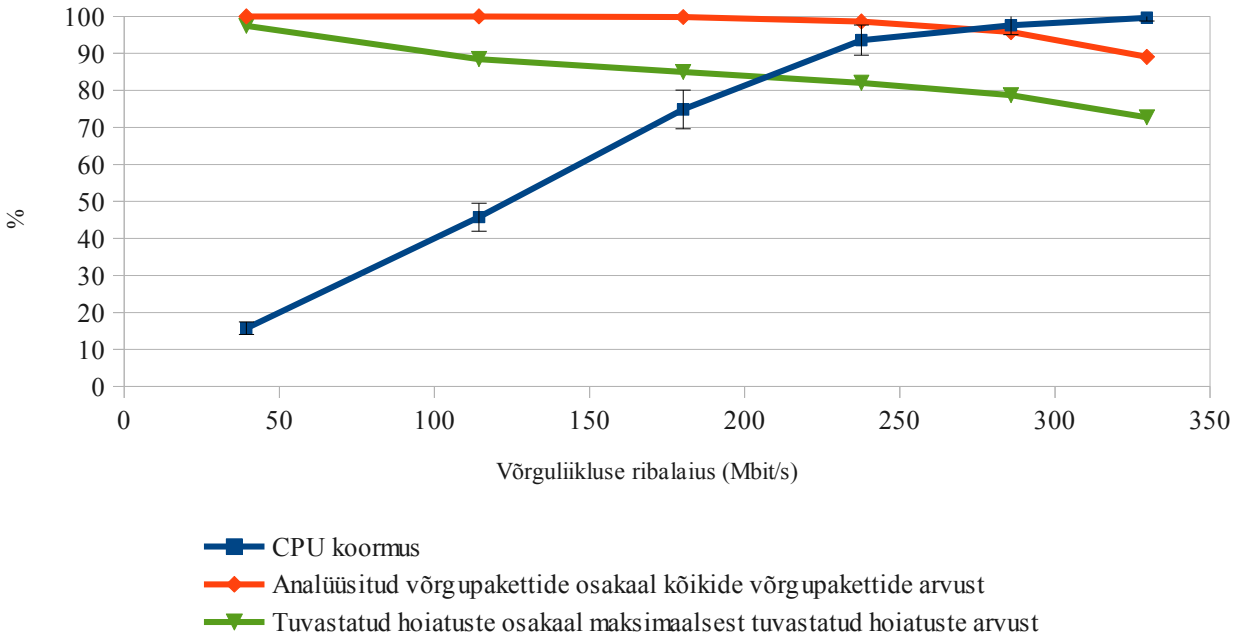
Joonis 8. Jõudluse mõõtmise implementatsioon 2: Suricata, PCAP, 1 CPU, OpenBSD.



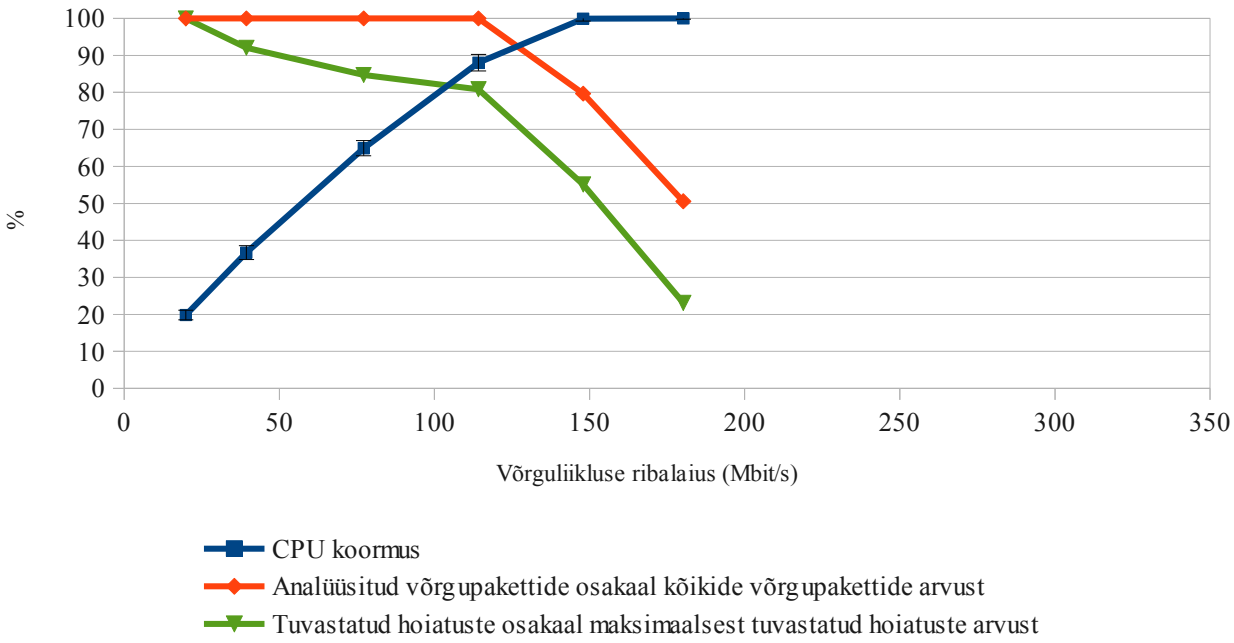
Joonis 9. Jõudluse mõõtmise implementatsioon 3: Suricata, PCAP, 2 CPU, OpenBSD.



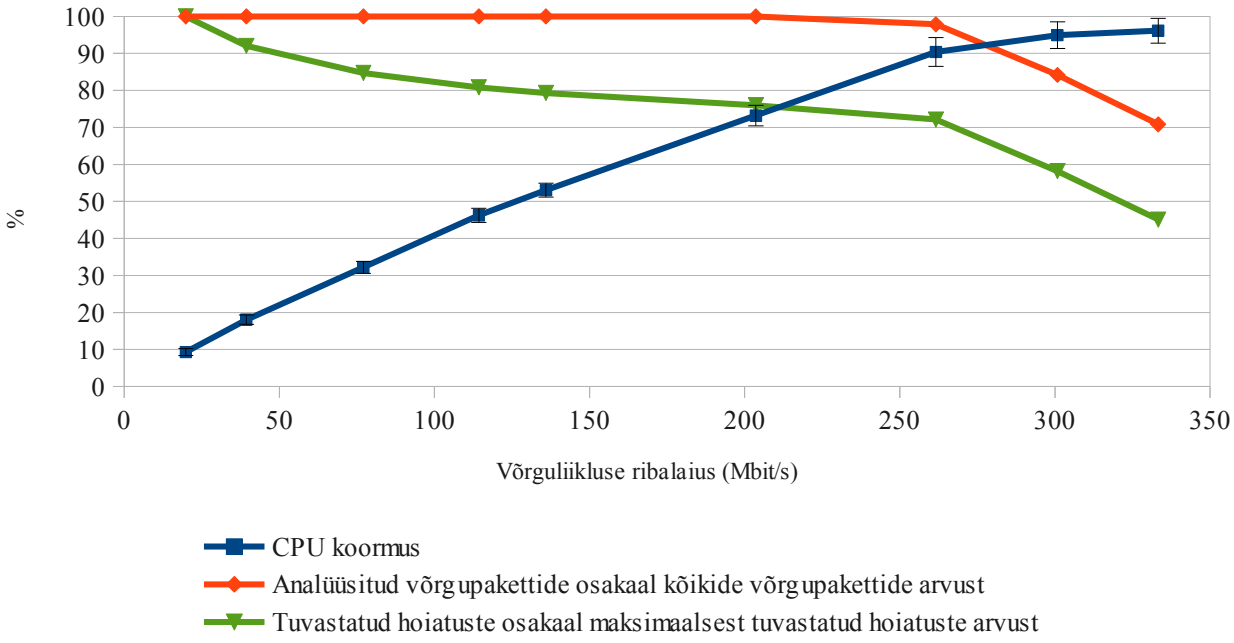
Joonis 10. Jõudluse mõõtmise implementatsioon 4: Snort, AF_PACKET, 1 CPU, Linux.



Joonis 11. Jõudluse mõõtmise implementatsioon 5: Snort x 2, PF_RING, 2 CPU, Linux.



Joonis 12. Jõudluse mõõtmise implementatsioon 6: Suricata, AF_PACKET, 1 CPU, Linux.



Joonis 13. Jõudluse mõõtmise implementatsioon 7: Suricata, AF_PACKET, 2 CPU, Linux.

Joonistelt saame järeldada, kas peatükis 2 püstitatud hüpoteesis omadused kehtivad:

1. Kõikide implementatsioonide korral kehtib, et võrguliikluse ribalaiuse kasvades kasvab CPU koormus.
2. Kõikide implementatsioonide puhul võrguliikluse ribalaiuse kasvades vähenevad analüüsitud võrgupakettide ja tuvastatud hoiatuste osakaal. Detailsemalt vaatleme neid eraldi:
   2.1. Analüüsitud pakettide arvu puhul kehtib välja pakutud omadus enamike implementatsioonide puhul, aga lõikumispunkt pole täpselt 100%, vaid natuke vähem. Snort puhul väheneb analüüsitud pakettide arv alla 98% alles siis, kui CPU koormus kasvab üle 97%. Suricata puhul hakkab analüüsitud pakettide arv varem vähenema. Kõige suuremaks erandiks on implementatsioon 3, kus analüüsitud pakettide arv hakkab CPU koormuse kasvades oluliselt varem vähenema. Sellest saame järeldada, et OpenBSD peal ei jaotu Suricata töö nii ühtlaselt lõimede vahel ära ning lõimede töökoormus ja sellest tulenevalt ka CPU koormus kõigub väga suures vahemikus. Implementatsioon 2 puhul saab märgata sama trendi, aga nõrgemalt.

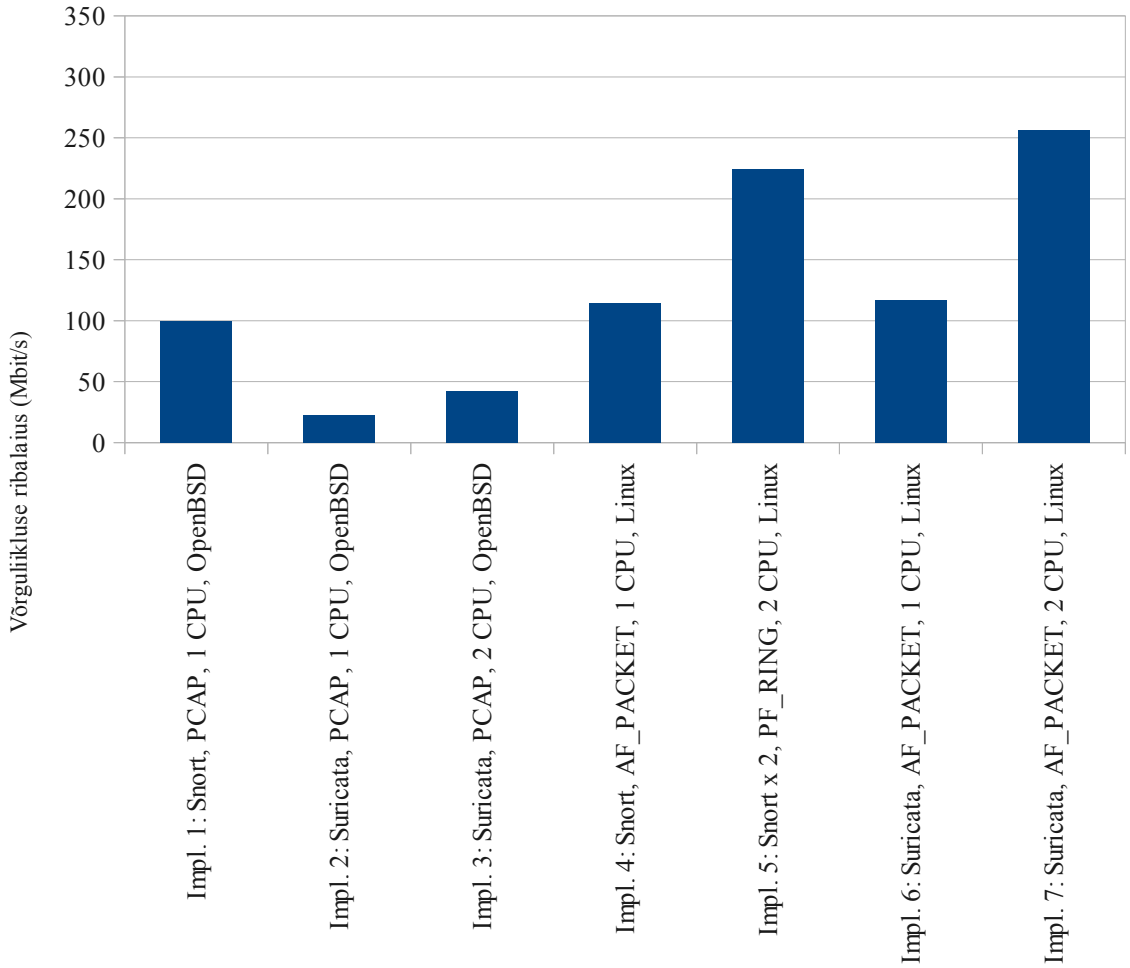
2.2. Hoiatuste arv ei püsi ühtlaselt kõrgel enne CPU koormuse 100%-ni jõudmist, vaid väheneb ühtlaselt koos võrguliikluse ribalaiuse ja CPU koormuse kasvuga.

3. Muutmälu hõivatus ei sõltu võrguliikluse ribalaiusest.

4 Võrdlus ja ettepanekud

4.1 Jõudluse võrdlus

Sissetungi tuvastuse süsteemi poolt raporteeritava analüüsitud võrgupakettide osakaalu näitaja jälgimine ning selle alusel süsteemi suutlikkuse hindamine on levinud praktika [22] [16] [17]. Järgnevalt kasutame analüüsitud võrgupakettide osakaalu implementatsioonide jõudluse omavaheliseks võrdlemiseks. Selleks leiame iga implementatsiooni puhul võrguliikluse ribalaiuse, alates millest langeb analüüsitud võrgupakettide osakaal alla 99%. Leitud võrguliikluse ribalaiuste võrdlus on toodud joonisel 14.



Joonis 14. Implementatsioonide jõudluse võrdlus.

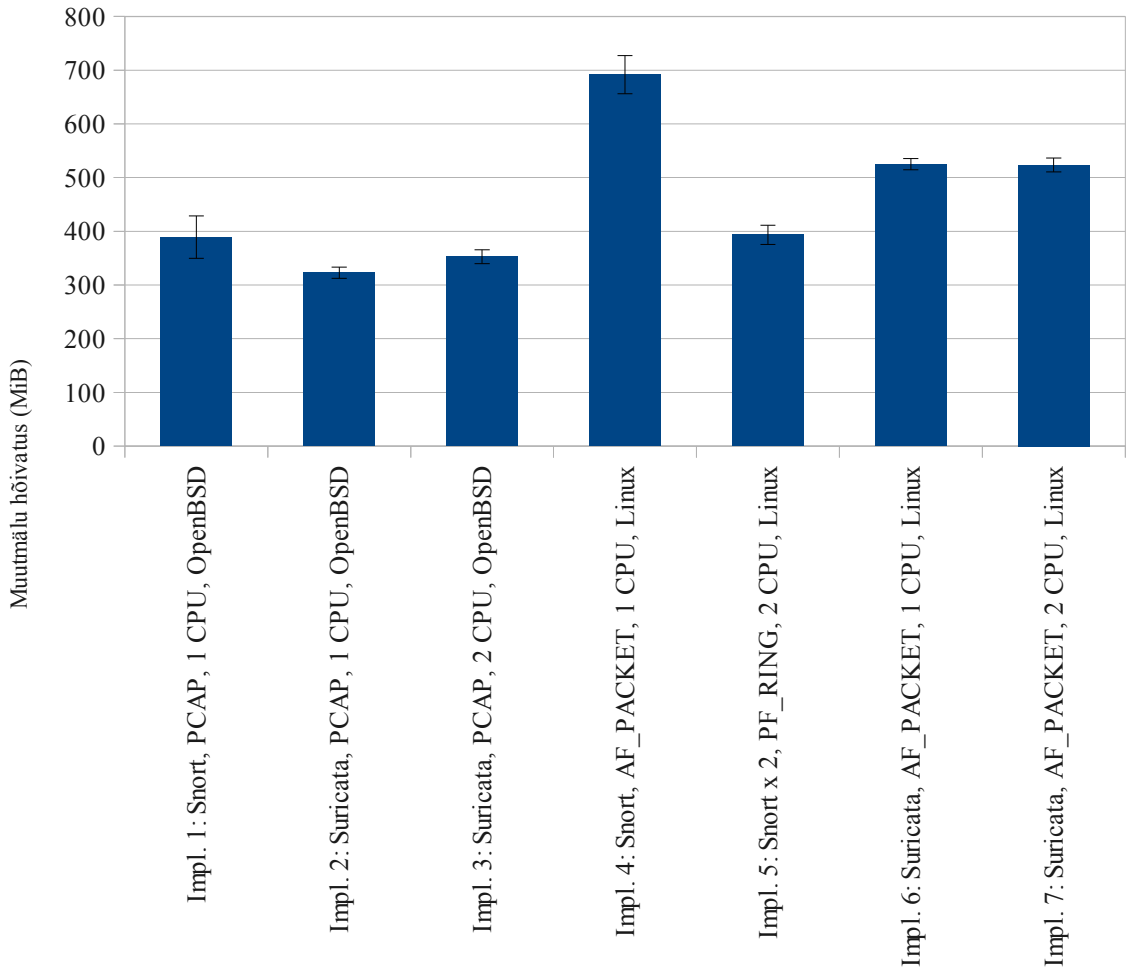
Järeldame, et:

- OpenBSD peal on Suricata jõudlus Snort'i omast oluliselt halvem, 4,5 korda 1 CPU puhul. Kui Snort kasutab 1 CPU ja Suricata 2 CPU'd, siis on Suricata jõudlus Snort'i omast samuti oluliselt halvem, 2,4 korda.
- Linux'il on Suricata jõudlus Snort'i omaga samas suurusjärgus ja isegi ületab seda kuni 1,1 korda.
- 1 CPU pealt 2 CPU peale vahetamisel muutub jõudlus samas suurusjärgus, 1,9-2,2 korda suuremaks.
- Snort'i jõudlus Linux'il on 1,1 korda suurem kui OpenBSD peal. Suricata jõudlus Linux'il on 5,2-6,0 korda suurem kui OpenBSD peal.

Varasemate tööde käigus on leitud, et Suricata jõudlus on mitu korda väiksem kui Snort'il [22] [17] [16]. Käesoleva töö käigus leidsime, et kasutatud seadistuste ning 5373 reegli korral on Linux'il Suricata tarkvara viimase versiooni jõudlus Snort tarkvara viimase versiooni omaga samas suurusjärgus ja isegi ületab seda minimaalselt.

4.2 Muutmälu hõivatuse võrdlus

Implementatsioonide muutmälu hõivatuse võrdlus on toodud joonisel 15. Käesolevalt jooniselt järeldame, et muutmälu hõivatus on sarnases suurusjärgus sama IDS ja võrgupakettide hõlmelahenduse kombinatsiooniga implementatsioonide puhul.



Joonis 15. Implementatsioonide muutmälu hõivatuse võrdlus.

Eelmises peatükis leidsime, et muutmälu hõivatus ei sõltu võrguliikluse ribalaiusest. Seega ei ole muutmälu hõivatus oluline kriteerium ettepanekute tegemiseks ja kogu süsteemi jõudlust suurendava lahenduse valimiseks.

4.3 Ettepanekud

S4A praegusele tüüpilisele juurutusele on kõige lähedasem implementatsioon 1 (Snort, PCAP, 1 CPU, OpenBSD). Kuigi S4A tüüpilise juurutuse riistvaral on 2 CPU tuuma, siis Snort kasutab neist ainult ühte, kuna on ühelõimeline. Tuvastasime, et sellisel juhul on teine CPU tuum on koormatud ainult kuni 2%, mis on piisavalt vähe, et seda järgneva võrdluse raames kõrvale jätta ning lugeda S4A praegune tüüpiline juurutus vastavaks implementatsioonile 1.

Peatükis 4.1 toodud jõudluse võrdluse alusel leiame, et kõige efektiivsemad viisid S4A süsteemi jõudluse suurendamiseks on järgnevad:

1. Võtta OpenBSD asemel kasutusele Linux ning kasutada PF_RING võrgupakettide hõlmelahendust koos CPU tuumade arvule vastava Snort protsesside arvuga (implementatsioon 4). Sellisel juhul saavad kõik CPU tuumad kasutatud ja ei toimu ressursi raiskamist. Tüüpilise S4A juurutuse puhul (2 CPU'd) tuleb jõudluse kasvuks kuni 2,2 korda. Omadused:
   a) Mitme Snort protsessi haldamine on natuke keerukam kui ühe Snort protsessi haldamine, mis suurendab kulutusi.
2. Võtta kasutusele Linux'i peal Suricata (implementatsioon 7). Sellisel juhul saavad samuti kõik CPU tuumad kasutatud ja ei toimu ressursi raiskamist. Tüüpilise S4A juurutuse puhul on jõudluse kasv esimesele ettepanekule omakorda lisaks 1,1 korda. Omadused:
   a) Ühe protsessi haldamine on vähem keerukam ja odavam kui mitme protsessi haldamine, mis vähendab kulutusi.
   b) Kulutusi suurendab see, et S4A realisatsiooni peab kohandama ning täiendavalt testima, kuna see tarkvara võib Snort'ist erinevalt käituda.

Alternatiivsed viisid S4A süsteemi jõudluse suurendamiseks, mis on vähem efektiivsemad kui ülaltoodud:

3. Suurendada riistvara jõudlust.
4. Vähendada kaudselt võrguliikluse mahtu, paigaldades mitu tuvastajat, igaüks eraldi alamvõrku, siis rakendub igale tuvastajale väiksem võrguliikluse maht.

Kokkuvõte

Käesoleva töö käigus tutvuti S4A süsteemi ülesehitusega ja Snort tööpõhimõttega. Viimase alusel analüüsiti selle jõudlust mõjutavaid tegureid. Leitud tegurite alusel pakuti välja erinevad võimalused S4A süsteemi jõudluse suurendamiseks. Detailsemaks analüüsiks kaaluti tarkvara implementatsiooni muutvaid võimalusi – vahetada operatsioonisüsteemi (OpenBSD, Linux), võrgupakettide hõlmelahendust (PCAP, AF_PACKET, PF_RING) ning tarkvara ennast (Snort, Suricata).

Analüüsimiseks valitud võimaluste kohta sooritati jõudluse mõõtmised erinevate võrguliikluse ribalaiuste juures, jättes muud tegurid nagu reeglite arv ja seadistused jt samaks. Mõõtmistulemused kinnitasid enne mõõtmisi välja pakutud hüpoteese, et võrguliikluse ribalaiuse kasvades kasvab CPU koormus ning võrguliikluse ribalaiusest ei sõltu muutmälu hõivatus. Mõõtmistulemused näitasid, et analüüsitud pakettide osakaal püsib ühtlaselt kõrgel seni, kuni CPU koormus püsib teatud piirist madalamal. Mõõtmistulemuste põhjal selgus, et tuvastatud hoiatuste arvu sõltuvus võrguliikluse ribalaiusest ei vasta püstitatud hüpoteesile. Nimetatud sõltuvuse lähem analüüs jäi praeguse töö eesmärkidest välja, kuid väärib kindlasti lähemat uurimist edasises töös.

Mõõtmistulemuste alusel loodi võrdlus, kus leiti, et mõõtmiste käigus kasutatud seadistuste ja 5373 reegli korral on OpenBSD peal Suricata jõudlus Snort'i omast oluliselt halvem; Linux'il on Suricata jõudlus Snort'i omaga samas suurusjärgus ja isegi ületab seda minimaalselt; 1 CPU pealt 2 CPU peale vahetamisel suureneb mõlema tarkvara jõudlus samas suurusjärgus; Snort'i jõudlus Linux'il on minimaalselt suurem kui OpenBSD peal; Suricata jõudlus Linux'il on mitu korda suurem kui OpenBSD peal.

Võrdluste alusel tehti järeldused ja ettepanekud S4A lahenduse jaoks. Esmane ettepanek on võtta OpenBSD asemel kasutusele Linux koos PF_RING võrgupakettide hõlmelahendusega ja masinas käivitada paralleelselt sama palju Snort protsesse, kui on masinas CPU tuumasid. See võib anda tüüpilise 2 CPU tuumalise S4A juurutuse puhul jõudluse kasvu 2,2 korda. Teine ettepanek on Linux'il kasutada Snort'i asemel Suricata't, mis annab

vähemalt sama suure jõudluse kasvu kui esimene ettepanek. Kummalgi lahendusel on omad positiivsed ja negatiivsed küljed, mida tuleks konkreetses süsteemis rakendamisel kaaluda.

A high-performance network intrusion detection solution for S4A software

Master's Thesis (30 ECTS)

Alar Kvell

Summary

S4A is a distributed real-time network security monitoring system. The purpose of this thesis is to find ways to increase S4A system performance.

S4A system consists of one center and several detectors. Detectors are computers that run specialized software and are installed into various government organization computer networks. All network traffic passing between the organization's internal network and public Internet needs to mirrored to detector machine's network interface in order network security monitoring to work.

Detector software consists of OpenBSD operating system, Snort intrusion detection system (IDS) and S4A specific helper programs. One of the most important qualities of an IDS is its performance – the more traffic it can analyze, the higher are chances of detecting suspicious traffic. Main bottleneck of current S4A system is that Snort is a single-threaded application and thus can only utilize one CPU core at a time, even though there are two CPU cores available in a typical S4A installation.

In this thesis we first give an overview of Snort's architecture and look at the properties that affect it's performance the most. For the most effective ways to increase its performance we consider the following: changing operating system to Linux; using an alternative network packet capture library on Linux to allow multiple simultaneous Snort processes so that all CPU cores would be utilized; using an alternative IDS Suricata instead of Snort. Then we measure each option's performance, replaying network traffic at different bandwidths. During these tests we observe CPU usage, memory usage, disk I/O

usage, analyzed packet count and detected alert count. We draw some conclusions on how they behave.

The main find is that using our configuration and detection rules, on Linux the latest version of Suricata has at least the same performance as multiple processes of the latest version of Snort, even slightly higher. On OpenBSD, Suricata performs much worse than Snort.

We make two suggestions for increasing S4A performance: 1) switching to Linux and multiple Snort processes with PF_RING may provide up to 2,2 times better performance; 2) switching to Linux and Snort should provide the same increase in performance as the first suggestion. Both options have some positive and negative effects that need to be considered before applying them.

Viited

[1] S4A is infrastructure for intrusion detection. http://code.google.com/p/s4a/ (viimati vaadatud 13.05.2012)

[2] Riigivõrgu turvarikete hajutatud jälgimissüsteem S4A. 2010. http://www.cyber.ee/infosusteemid/dokumendid-failidena/S4A_infoleht_2010.pdf (viimati vaadatud 13.05.2012)

[3] OpenBSD kodulehekülg. http://www.openbsd.org/ (viimati vaadatud 13.05.2012)

[4] Snort kodulehekülg. http://www.snort.org/ (viimati vaadatud 13.05.2012)

[5] TCPDUMP/LIBPCAP kodulehekülg. http://www.tcpdump.org/ (viimati vaadatud 13.05.2012)

[6] Dell. Dell PowerEdge R200 server. 2010. http://www.dell.com/downloads/global/products/pedge/en/pe_R200_spec_sheet_new.pdf (viimati vaadatud 13.05.2012)

[7] Definition Monday: Intrusion Detection Systems. 2011. http://www.digitalundercurrents.com/blog/?p=82 (viimati vaadatud 13.05.2012)

[8] Snort, IDS, IPS, NSM, hacking and beyond. 2009. http://harrykar.blogspot.com/2009/05/snort-ids-ips-nsm-andbeyond.html (viimati vaadatud 13.05.2012)

[9] The Linux Kernel Archives kodulehekülg. http://www.kernel.org/ (viimati vaadatud 13.05.2012)

[10] PF_RING High-speed packet capture, filtering and analysis kodulehekülg. http://www.ntop.org/products/pf_ring/ (viimati vaadatud 13.05.2012)

[11] The Open Information Security Foundation kodulehekülg. http://www.openinfosecfoundation.org/ (viimati vaadatud 13.05.2012)

[12] Parallel Programming and Computing Platform CUDA kodulehekülg. http://www.nvidia.com/object/cuda_home_new.html (viimati vaadatud 13.05.2012)

[13] Nigel Jacob, Carla Brodley. Offloading IDS Computation to the GPU. Computer Security Applications Conference, 2006. ACSAC '06. 22nd Annual, pp 371-380, 2006.

[14] Giorgos Vasiliadis, Spiros Antonatos, Michalis Polychronakis, Evangelos P. Markatos, Sotiris Ioannidis. Gnort: High Performance Network Intrusion Detection Using Graphics Processors. Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection, Volume 5230/2008, pp 116-134, 2008.

[15] Winston H. Messer. Performance Testing Suricata: The Effect of Configuration Variables On Offline Suricata Performance. 2011. https://redmine.openinfosecfoundation.org/attachments/download/706/Messer-Practicum-Final-v4.pdf (viimati vaadatud 13.05.2012)

[16] Albin, E.; Rowe, N.C.. A Realistic Experimental Comparison of the Suricata and Snort Intrusion-Detection Systems. Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on, pp 122-127, 2012.

[17] David J. Day, Benjamin M. Burns. A Performance Analysis of Snort and Suricata Network Intrusion Detection and Prevention Engines. Fifth International Conference on Digital Society, Gosier, Guadeloupe, pp. 187-192, 2011.

[18] Adeeb Alhomoud, Rashid Munir, Jules Pagna Disso, Irfan Awan, A. Al-Dhelaan. Performance Evaluation Study of Intrusion Detection Systems. The 2nd International Conference on Ambient Systems, Networks and Technologies, volume 5, pp 173–180, 2011.

[19] Vít Prajzler. Network Traffic Analyzer and Generator for FastEthernet. Czech Technical University, 2010. https://dip.felk.cvut.cz/browse/pdfcache/prajzvit_2011dipl.pdf (viimati vaadatud 13.05.2012)

[20] Tcpreplay koduleheküIg. http://tcpreplay.synfin.net/ (viimati vaadatud 13.05.2012)

[21] Emerging Threats.net Open rulesets koduleheküIg. http://rules.emergingthreats.net/open/ (viimati vaadatud 13.05.2012)

[22] Mike Lococo. Understanding Snort Performance: An Evidence-Based Approach. 2011. http://mikelococo.com/files/2011/2011_01_25-snort_performance.pdf (viimati vaadatud 13.05.2012)

Lisad

Lisa 1. Tuvastaja keskkonna info

Järgnevalt toome välja tuvastaja masina riistvara ja tarkvara info, et mõõtmiskeskkonda ja mõõtmistulemusi oleks võimalik reprodutseerida. Võrguliikluse mahamängimise masina infot toodud ei ole, kuna selle puhul on oluline vaid võrguliikluse saatmise ribalaiuse jäljendamine, mida saab tagada ka erineva riistvara ja tarkvara korral.

Riistvara:

- Protsessor Intel Core2 Duo E8200 2.66GHz (2 tuuma);
- Emaplaat Intel DP35DP;
- Muutmälu 2 x DDR2 800 MHz, 2 GiB;
- Võrgukaart Intel PRO/1000 82566DC-2.

Operatsioonisüsteemid:

- Debian 6.0 amd64 arhitektuuril (kõik pakid uuendatud 05.05.2012 seisuga);
- OpenBSD 5.1 snapshot 26.04.2012 amd64 arhitektuuril.

Autori poolt kompileeritud tarkvara:

- Snort 2.9.2.2 ja DAQ 0.6.2 (välja lastud 27.03.2012);
- Suricata 1.3beta1 (välja lastud 04.04.2012);
- libpcap 1.2.1;
- PF_RING 5.2.1;
- libdnet 1.12 (Linux'il).

Kompileerimisel kasutatud keskkonnamuutujad:

- CFLAGS="-O2 -pipe -march=native"
- CXXFLAGS="${CFLAGS}"
- CPPFLAGS="${CFLAGS}"

- LDFLAGS="-Wl,-O1"

Operatsioonisüsteemides oli mõõtmise ajaks saalemälu (*swap*) välja lülitatud. Samuti olid üleliigsed teenused suletud, nii et enne mõõtmise sooritamist töötasid ainult järgmised protsessid:

- init
- gkrellmd
- rsyslogd (Linux'il) / syslogd (OpenBSD'l)
- ntpd
- sshd
- getty
- udevd (Linux'il)
- acpid (Linux'il)

Mõõtmise sooritamise ajal töötasid lisaks ülal loetletud protsessidele ja Snort / Suricata protsessile ka järgnevad protsessid:

- dstat (Linux'il);
- ifstat ja vmstat (OpenBSD'l).

Reeglite allikaks oli Emerging Threats open ruleset 7090, kuupäevaga 09.05.2012.