UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

OPEN SOURCE IDS/IPS IN A PRODUCTION ENVIRONMENT: COMPARING, ASSESSING AND IMPLEMENTING

MESTRADO EM SEGURANÇA INFORMÁTICA

João Paulo da Costa Calado

Trabalho de projeto orientado por:
Prof. Doutor Hugo Alexandre Tavares Miranda
e co-orientado pelo Lic. Pedro Miguel Raminhos Ribeiro Botas

2018

This page intentionally left blank.

Acknowledgments

I would like to start by thanking both Pedro and professor Miranda. Thank you for the opportunity of doing something that I like, but most importantly where I find challenge. Thank you for believing in Free/Libre and Open Source Software, I really believe we are making a difference with it. A big ”THANK YOU” to the teachers I found along the way, I don’t think they’ll ever know how they touch our lives and how deeply we keep their teachings with us. I will miss this interaction. Thanks to professor João Ascenso, he is definitely the person responsible for the path that my career and studies have taken, the one who awakened my interest for such topics. Would like to thank my colleagues from school, the ones that, for the last couple of years, shared their time, their knowledge, their experiences, their strength in despair, courage and determination. The ones that did not throw in the towel, and the ones that do not dare to consider it. My colleagues at work, with whom I can always count to take the discussion a step further, to keep me challenged and to obtain both personal and professional guidance. I am grateful to everyone who has believed and supported me for the last couple of years and throughout this path. It has been a long journey and every little action and encouraging word meant the world to me. Acima de tudo, obrigado à minha família: aos meus avós, aos meus pais e ao meu irmão; à minha mãe, que sempre acreditou que chegaria a bom fim. À Sara, que um dia tomou a decisão(?) de me acompanhar ”nesta história”, sem ela nada haveria para contar. À Maria e à Inês, as minhas pequenas egérias; porque nunca conseguirei devolver-vos o tempo que vos tirei.

To one of the most dedicated, committed and passionate teachers that I have ever met, José Fernando Duarte do Amaral, thank you so much for being a part of my path. May your name never be forgotten.

This page intentionally left blank.

in memory of the Ian in Deb'

This page intentionally left blank.

Resumo

Este trabalho descreve a concretização de uma solução IDS (*Intrusion Detection System*) num ambiente produtivo e de alta disponibilidade. Pretendeu-se avaliar a sua exequibilidade e nível de confiança nos resultados, comparando algumas opções e abrindo assim a hipótese de colocar esta solução em modo *inline*. Desta forma, em conjunto com uma implementação de *iptables*, poder-se-á considerar a substituição de uma atual solução de segurança, de *hardware* e *software* proprietário, por uma antepara de segurança e IPS (*Intrusion Prevention System*) de Software Livre ou de Código Aberto (*Free/Libre and Open Source Software (FLOSS)*).

Tipicamente, o mercado apresenta produtos desenvolvidos para este efeito recorrendo a *hardware* dedicado, criando caixas negras, altamente eficientes e robustas. Para estes produtos, os fabricantes garantem uma série de compromissos, no entanto, tirando partido de altíssimos valores pelo licenciamento, recursos e características adicionais ou mesmo até pelo suporte ao produto.

Por vezes estes produtos têm por base projetos da comunidade, sendo levados ao mercado por fabricantes em variantes proprietárias. Nesta perspetiva pretendeu-se, neste trabalho, avaliar a possibilidade de criar um ambiente de defesa inteiramente baseado em alternativas aos fabricantes, desde o sistema operativo até às camadas de avaliação do nível aplicacional.

De entre estes produtos, focámos o nosso trabalho em NIDSs (*Network Intrusion Detection Systems*), concretamente, analisando implementações de Snort e Suricata. Uma implementação deste género requer atenção a várias especialidades e à sua interoperabilidade, não esquecendo ainda os recursos físicos bem como os requisitos e características da operação dos utilizadores na rede. Procurou-se entender se, usando *hardware* comum, vulgarmente disponível no mercado, seria possível construir uma solução à semelhança de conjuntos de equipamento dedicado e construídos especificamente para o efeito.

Como fase inicial, procurou-se enquadrar a problemática olhando ao trabalho dos últimos 7 anos, neste campo científico. Este tema é revisitado continuamente, uma vez que os produtos estão em constante evolução. Foram revistas diversas comparações entre os dois produtos em análise, criando uma base sólida para o arranque da nossa investigação, delimitando, assim, melhor o foco.

Ainda que não sejam a parte elementar deste trabalho, incluíram-se algumas noções e

observações acerca de pormenores a considerar em trabalho futuro ou na passagem a produção desta solução. Temas como a integração com uma antepara de segurança em *software*, a passagem do tráfego desta para o sensor, e as limitações na aquisição de pacotes entre cartas de rede, núcleo do sistema operativo e sensor, foram analisadas para enquadrar e agilizar o trabalho futuro.

Já dentro da envolvente das aplicações, avaliou-se o modo destas serem alimentadas com assinaturas atualizadas, por forma a manter o sistema preparado para reagir de forma atempada a novos eventos de segurança. Ainda sobre este tema, realçou-se a importância dos recursos humanos envolvidos na gestão e manutenção do sistema, uma vez que requer uma contínua interação, de forma a manter os níveis de segurança e a reduzir os avisos de falsos positivos. Não menos importante, o trabalho aborda ainda as diversas opções de alarmística e registo de eventos. Neste sentido, avaliou-se um coletor de registos e correlacionador de eventos (SIEM) [*Security Information and Event Management (SIEM)*], para receber os registos do sistema IDS/IPS e acrescentar-lhes mais uma forma de inspeção. Estes produtos (SIEMs) permitem correlacionar eventos de várias fontes, entre eles, ou com as suas próprias assinaturas, detetando e alertando para eventos na rede, que possam ter despistado a análise de uma componente específica desta. Assim, conseguiu-se colmatar um dos requisitos apresentado, que visava a interpretação ágil e dinâmica dos alertas gerados pelo sistema IDS/IPS.

O trabalho contemplou uma série de simulações em laboratório (recorrendo a virtualização), a colocação em pré-produção da solução IDS, a comparação de resultados reais com tráfego real, retirando ainda a avaliação física de dimensionamento de recursos equiparáveis.

Em laboratório, foi possível reproduzir os diversos enquadramentos e *softwares* a explorar, bem como alternar ligeiramente os recursos físicos afetos aos sistemas, projetando desta forma as restantes fases do trabalho. Seguiram-se uma série de avaliações em modo *offline*, baseadas em ficheiros de capturas de pacotes (PCAPs) [Packet Capture]. Estas avaliações incluíram mais de 1500 testes, sendo que apenas 469 foram considerados úteis devido à sua dimensão. Concluiu-se que testes com capturas demasiado pequenas geravam sessões de apenas um ou poucos segundos, não permitindo assim intervalos para recolha de dados para análise. Estes testes foram efetuados num total de 4 ambientes diferentes, incluindo equipamento doméstico, equipamento industrial (em análise para implementação em produção) e dois ambientes de virtualização baseados em *cloud computing*, variando os seus recursos físicos. Entre outras conclusões, esta abordagem permitiu ressalvar as problemáticas inerentes caso uma decisão pendesse sobre uma implementação virtualizada. No entanto importa referir que o *hardware* doméstico teve a sua implementação assente em KVM (*Kernel-Based Virtual Machine (KVM)*) sem que tenham sido detetadas limitações semelhantes às detetadas em *cloud computing*. Neste enquadramento os resultados mostraram ainda que, a memória não parece ser uma limitação.

Ainda que está área pudesse ter sido melhor explorada, verificou-se que o consumo de memória diminui quando se aumenta o número de núcleos (CPU) [Central Processing Unit]. No entanto, (com o Suricata) a partir de determinado número de núcleos o consumo de memória volta a aumentar, ainda que ligeiramente.
Ainda nesta fase da investigação, adicionou-se a versão 3 do Snort (ainda em desenvolvimento) e efetuaram-se testes em ambos os ambientes fisicamente disponíveis. Detetou-se que, apesar desta versão, por natureza da sua arquitetura, usar múltiplos processos, a variação da disponibilização de mais e mais núcleos não altera o seu débito. Mas não havendo segurança numa versão ainda em desenvolvimento não se continuou esta linha de investigação.
Já em pré-produção, variaram-se configurações de alto desempenho em ambos os *softwares* uma vez que o esforço avaliado era elevado. Tanto no Snort como no Suricata, testaram-se técnicas de captura de pacotes usando métodos baseados em AF_PACKET e PF_RING, depois de algumas alterações efetuadas ao nível das cartas de rede. Testaram-se ainda técnicas de afinação relativamente a pré-processadores de análise e afetação de memória dedicada a estes. Foi possível ainda avaliar o comportamento do software quando executadas mais do que uma instância em simultâneo, sendo possível dedicar cada uma destas à inspeção do tráfego escutado em *interfaces* de rede separados. Assim, provou-se ser possível manter monitorizados e registados vários segmentos de rede, inspecionados de forma independente, sendo certo que o hardware permitia ainda adicionar mais carga, tanto em termos de processamento como de *interfaces* disponíveis.
Desta forma, apresentou-se, uma avaliação desta solução, para que seja tomada uma decisão informada acerca da sua possível implementação em produção, em substituição de uma solução proprietária.
Neste trabalho, provou-se que, de facto, é possível usar *hardware* comum para implementar tal solução no ambiente testado e com o esforço de tráfego apresentado na prova de conceito. Pelo menos um dos IDSs testados (Suricata) funcionou de forma infalível, por vários dias, num ambiente de rede altamente denso e complexo. Neste caso específico, o IDS atuou continuamente, sem perdas registadas, num enquadramento onde por vezes foi possível medir mais de 3Gbps, com picos por volta dos 4,5Gbps. Adicionalmente, foi possível executar duas instâncias em simultâneo, com cada uma inspecionando um *interface* de rede, dedicado, de 10Gbps. Deixaram-se ainda notas importantes a considerar na possível colocação deste sistema em produção, ressalvando aspetos relativos aos recursos humanos, bem como aos recursos físicos e sua interoperabilidade. Será conveniente não descurar um período de pré-produção, adaptação das regras e customização das configurações que melhor se adaptem à concretização final. Importante ainda, a ter em consideração, a forma de escalamento destes sistemas, que pela natureza das suas limitações poderão apenas expandir partilhando o esforço com outras unidades análogas.

Palavras-chave: IDS, IPS, Snort, Suricata

This page intentionally left blank.

Abstract

This work describes the realization of an IDS solution in a productive environment. It was intended to evaluate its feasibility comparing some options and thus opening the possibility of putting this solution in inline mode. Hence, the host organization may consider replacing a current security solution (proprietary hardware and software), with a Free Software or Open Source firewall and IPS.
Typically the market presents products developed for this purpose using dedicated hardware, creating highly efficient and robust black boxes. For these products the manufacturers guarantee a series of commitments, taking advantage of high values for licensing, additional features or even product support.
Sometimes these products are based on community projects being brought to market by vendors in proprietary variants. In this perspective, it was intended, in this work, to evaluate the possibility of creating a defense environment entirely based on alternatives to the manufacturers', from the operating system to the application's level evaluation layers.
This work provides a series of laboratory simulations (using virtualization), the placement in staging of the IDS solution, the comparison of actual results with real traffic, and retrieving the physical evaluation of comparable resources. In this way an evaluation of this solution will be presented to the host organization so that an informed decision is made about its possible implementation in production, to replace a proprietary solution.
We found that, in fact, it is possible to use commodity hardware to implement such solution in the tested environment, and with the presented traffic demand. At least one of the tested IDSs (Suricata) performed flawlessly, for several days, in a highly dense and complex network, where more than 3Gbps with peaks around 4.5Gbps were observed. The work also reports on scenarios where two concurrent instances were run, with each one inspecting a dedicated 10Gbps listening interface.

Keywords: IDS, IPS, Snort, Suricata

This page intentionally left blank.

Contents

This page intentionally left blank.

List of Figures

This page intentionally left blank.

List of Tables

This page intentionally left blank.

Chapter 1

Introduction

1.1 Motivation

Typically the market presents products developed for network security in dedicated hardware, creating highly efficient and robust black boxes. For such products the manufacturers guarantee a series of commitments, additional features and resources or even product support, while charging high amounts for licensing. Some of these security products are based on community-developed software, being brought to market by manufacturers in proprietary variants. In this work, we intended to evaluate the possibility of creating a defense environment based entirely on alternatives to manufacturers', from the Operating System (OS) to the application's level evaluation layers. The intention was also to consider the replacement of a current firewall solution with a Free/Libre and Open Source Software (FLOSS), complemented with an Intrusion Detection System (IDS); ideally this system should be allied to a firewall to serve as an Intrusion Prevention System (IPS). This work intended to prove the feasibility of this implementation in a reliable, tested and proven way. This work also intended to add new functionalities, similar to those on licensed products, that the market offers and which the Industry currently considers to be good practices, essential to ensure safety levels where risks are reduced and actions are taken to prevent or mitigate cyberattacks. Since some previous tests were done in the past by this department with a particular IDS, we aimed to thoroughly test back that IDS (Snort) in its more recent version, plus adding another well known Open Source Software (OSS) solution (Suricata).

1.2 Overview

An **Intrusion Detection/Prevention System** monitors events in an environment and analyzes them for possible threats against security policies. As the names state, the Detection system detects such threats in the events, while the Prevention system adds actions to stop such threats. Moreover, such systems can be drilled down in other *IDS such as HIDS,

NIDS or even WIDS. A Host Intrusion Detection System [Host-based IDS] (HIDS) monitors a single host and only the events related with it or the incoming and outgoing connections with it. The Network Intrusion Detection System (NIDS) monitors the entire traffic on a network for suspicious activities, within it or incoming/outgoing from it. Others, such as a Wireless Intrusion Detection System (WIDS), as its name states, works on Wireless related networks and protocols. Typically a NIDS relies on itself or rather in subsets of **sensors**, distributed across relevant segments of network, listening, capturing and monitoring the traffic. Detection can be either processed at these sensors or at the centralized facility. These sensors can be deployed either inline or in passive mode. A sensor in **passive mode** works as an IDS, listening to a copy of the traffic which is sent to it to process, by this not having direct interaction with it. **Inline mode** implies that the traffic passes through the sensor, typically working as an IPS combined with a firewall. A **firewall** is a security system which acts on the incoming and outgoing traffic passing through it. Such system will allow or deny traffic, based on predefined security rules and policies. A firewall is distinct from an IPS because it is rule based and analyzes packets' headers for addresses and ports only, an IPS analyzes both the header and the payload based on known events, patterns, behavior and signatures. A **log collector** is a system that collects and stores events' logs from several systems across a network. These systems allow to analyze and perform advanced and centralized investigation on available data. Moreover a **Security Information and Event Management (SIEM)** system combines Security Information Management (SIM) and Security Event Management (SEM) with log management, adding Security Event Correlation (SEC), to provide real-time security event's analysis.
In our work we are focusing on NIDS and we will refer to it simply as IDS or IPS where relevant. We addressed the topic against two specific products in this field, Snort and Suricata. Yet we added some experiments with Snort3 (project also known as Snort Alpha and sometimes Snort++). For clarity and simplicity, in this text we will refer to Snort2 as Snort; except where relevant or when Snort2 is discussed side-by-side with Snort3.

1.3 Objectives

The main objective of this work was to prove and evaluate the feasibility of this implementation so that the management would make an informed decision on the renewal or change of the actual security system in the upcoming months. The intention was to start this work in a phased way, implementing and testing the various parts of the system and making it more mature and robust in each phase, taking into account the priorities and feasibility of the deliverables. The following is a brief summary of the activities:

1. Simulations and laboratory evaluations.

2. Simulations and offline evaluations in ”production equivalent” hardware environments (using packets’ captures).

3. Implementation, demonstration and evaluation using mirroring (Intrusion Detection System (IDS)).

4. Demonstration of administration and management capabilities (interface or Security Information and Event Management (SIEM) integration).

With this work we sought to understand how commodity hardware would perform using either Snort or Suricata in replacement of vendor’s dedicated appliances and special hardware. We wanted to understand if it would be possible to deploy a complete state-of-the-art security solution entirely based on FLOSS and using common hardware; how to measure it and how to assess its scalability. We focused on core performance and tuning and not on rulesets as those would need to be adapted and customized against a specific environment and that takes time to tackle in complex networks such as the one we have worked on. In that sense we relied on pre-existing sets, provided and maintained up-to-date, to kickoff our work.

1.4 Contributions

In our work we have reviewed several years of comparisons using distinct approaches, between these two softwares. We provide an up-to-date comparison for the mid 2018’s versions of the products, where we compared its performance across several types of environments, in search of possible deployment scenarios, its scalability and caveats for implementation. We addressed our intention of relying on Free/Libre and Open Source Software (FLOSS) only, and also added some additional hypervisor possibilities for comparison. We have built step-by-step documentation, to provide guidance for the implementation, if approved for deployment, or either to publish it within the relevant communities. We tried to include the newest Alpha version of Snort version 3, which would have given us a nouvelle set of information on its performance, since there is a reduced amount of information published in that matter. As an extra mile, we’ve included concerns and details to be addressed ”before” and ”after” the sensor’s deployment in the traffic’s pipe, including concepts on pipping from the firewall, packet acquisition, outputs, log’s and alerts’ facilities piping to external log collectors and event correlation engines. We provide pre-production results, taken from a highly dense and complex network such as a university, with thousands of concurrent users, allowing to identify how common hardware performs with such demand.

1.5 Document structure

This document starts by providing a summary of investigation works from the last 7 years in this matter. It follows with the analysis of the presented problem, how such tools work and interact before and beyond its position in a network, or as a security system as a whole. Next, we present the design of the solution, starting from its implementation in laboratory and moving to foreseen details for final implementation and deployment in production, the offline approach to test agnostically across hardware (or virtualized) possibilities and finally, the pre-production setup using real traffic. Next, we present the implementations' details, testbeds used, techniques and additional software to support the work. Also, we summarize our findings of the gathered results, both, in offline mode and in pre-production. Finally we present our conclusions leaving some topics for future work and/or to be checked while moving this setup to production. We finish the document with appendices describing the softwares' setups, some troubleshooting steps, common usage and commands; and lastly, the full extent of the relevant plots from the data collected in offline mode.

Chapter 2

Related work

The topic addressed in this work has been the focus of much discussion in the industry as well as in academia. It is becoming increasingly clear that security levels in infrastructures must, not only, be restrictive but also dynamic. Networks (especially perimeter's) must be secured by systems with autonomy and intelligence. This autonomy and intelligence is guaranteed by the typical system training and also by centrally available subscriptions, fed and made available by manufacturers or the community. The proposed work is intended to meet the specific requirements of an implementation as a whole, and it is therefore necessary to evaluate its components as part of the system. In an initial phase, and for a global view of the system, we've consulted the work of Jorge Granjal [18] describing comprehensive approaches to security problems and solutions, with practical implementations of those solutions, based on Linux. With a clear path for the intended approach the work focused on the comparison of the IDS component and therefore we dug for previous works.

In 2011, with Suricata released less than 2 years before, Eugene Albin [2] compared both Snort and Suricata looking for speed, memory and accuracy figures. Suricata was adding multi-threading to the state-of-art for IDSs, and as such his work concluded that Suricata could handle larger volumes with similar accuracy. Though his work was performed on a busy virtual environment he had access to a network pipe of 20Gbps of bandwidth with an average of 200Mbps per day.

Also in 2011, Jonas Taftø Rødfoss [38], with a similar intention, added Bro to the comparison. He focused on alarms and logs with normal and triggered traffic and also on the installation processes. He concluded that his approach with Bro and using Metasploit was time consuming to configure. Rødfoss also tried to run simultaneous instances and found some issues, therefore he opted for offline Packet Capture (PCAP), providing him a replicable scenario. In one of his experiments, with a 4 days PCAP, he got 40GB that Bro was able to process in almost 28 minutes, Snort in less than 54 minutes and Suricata in

4 hours and 44 minutes. A second verification for Suricata took 4 hours and 47 minutes. This was attributed by Rødfoss to problems for Suricata processing PCAPs. Nonetheless he also found that Suricata was much easier to install and configure, less dependencies issues as those found for Snort. He also found a lot of spam alerts with Snort and Bro.

Already in 2012, Mauno Pihelgas [32], compared all three above-mentioned IDSs, looking for advantages and disadvantages of each one. Evaluating in a 1Gbps network, he found that all could handle 100Mbps, tested with PF_RING (a high-speed packet capture network socket) at 1Gbps with no drops. His focus was on drops and its optimization to achieve performance beyond 100Mbps. With Snort he was able to achieve 450Mbps with AF_PACKET module (an interface optimized for high performance packet processing) and no possible results with PF_RING, just unresolved errors; though with Suricata and Bro he was able to achieve 1Gbps with no drops using PF_RING.

In 2013 Joshua White et al. [12] tested both Snort and Suricata for performance with comprehensive quantitative comparison, based on Snort's developers arguments that Suricata, with multi-threading, would be slowing down the detection as system resources are scaled up. They've concluded that Suricata, with single thread, was faster than Snort where they would expect Snort to perform better. They also concluded that Suricata would have problems with high scalability. For this work a methodology was created for testing, and made available to be extended with additional PCAPs, additional rules and additional configurations or hardware platforms and different environments. They focused their metrics on PPS, RAM and CPU values for the process being run. They took Packets per Second (PPS) from built-in stats provided by the IDS at 1 second intervals and remaining metrics from the OS perspective, using PS commands and script parsing to achieve the same readings for both the System Under Test (SUT)s. In this work they've shared their scripts and files encouraging others to enlarge findings.

Still in 2013 R. China et al. [15] focused on packet drops which they found to increase with the network's throughput, also if packet size increases (tested with 512 and 1024 bytes) drops decrease. Both Snort and Suricata performed with the same behavior with larger packets and larger throughputs, but they found Snort to be more stable in lower demand.

In 2015 a new work looked into more modern bandwidth values, up to 10 and potentially 40Gbps, focusing on the importance of scaling and future problems. This work by George Khalil [24] also found libpcap to be limited so they moved to AF_PACKET. He found Suricata and Bro to introduce GeoIP lookups and Suricata introducing the possibility for GPU acceleration and also IP reputation. Regarding Snort he looked into it

introducing multi-instance support. This work highlighted that networks with 100Gbps are expected to be cost effective in a few years and as such sensor's CPUs won't be able to keep up with the demand, what leaves the necessity to approach scenarios with multiple CPUs or multiple sensors sharing the load. Khalil also highlighted that with more and more (client) systems in the network, system's patches, updates and maintenances tend to modify behaviors and can generate false positives or trigger IPSs automatic denials/drops, which lead to the necessity of an appropriately sized and experienced team to maintain and respond to events generated.

Works in 2016 commented on Suricata's performance over Snort's, based on the multi-threading characteristics [13] and tests using pytbull on HIDS mode [16], they've concluded again that Snort performs well at low requisites and that Suricata presents more features and more performance, apparently being more future proof. For example, allowing to move computation to GPU as we see more and more in other sort of applications. Later in 2016 the same author [14] presents an approach for Snort, adding a preprocessor for anomaly detection based on profiling.

Finally in 2017 Resmi [37] published a high level survey of different IDSs and Gunadi [19] a descriptive comparison of characteristics of the before-mentioned three IDSs.

This page intentionally left blank.

Chapter 3

Analysis

Considering the goals to deploy the solution in-line, acting as an IPS, we needed to understand how the system would perform as an IDS beforehand. The defense system should neither allow packets to pass unchecked, missing possible suspicious activity, nor bottleneck, dropping packets because it is overwhelmed by the amount of workload. Looking at previous works, and to the products' documentation, it is clear that rules will need to be customized according to the deployed environment, but that will also be truth for the firewall part of the system. Therefore the focus of this work aimed to find a non-packet-dropping NIDS, deployed to accommodate the host organization's network loads, based on its performance rather than its reactions and inspections results. Because these inspection's results will be dependent and dynamic, determined by the network segment that will be inspecting and acting upon. We sought for Packets per Second (PPS), CPU and RAM metrics, to understand how would the available hardware behave and if it would be suitable to perform as it would a dedicated physical appliance.

3.1 Firewall

Though we are not presenting in-depth implementation details of iptables in this work, we are including this short section on how iptables interact with the IPS. As in Figure 3.1, for Snort's inline mode, iptables can be set to target userspace software through NFQUEUE. The listening application will later inspect and react according with its configured behavior. As seen in Figures 3.2, 3.3 and 3.4 Suricata allows the configuration of this behavior in several modes with different reaction types. The packet would either be processed by the IPS only (no more iptables rules involved), marked and re-injected back to iptables or routed to another tool [47]. In case of complex or highly customized iptables' rules sets nftables might be considered a better option, allowing a more granular rule deployment while still in the firewall. Also nftables implements techniques for CPU load balancing which can add yet another performance switch to the system.
Moreover, and because this topic is not covered by this work, it is advised that in future

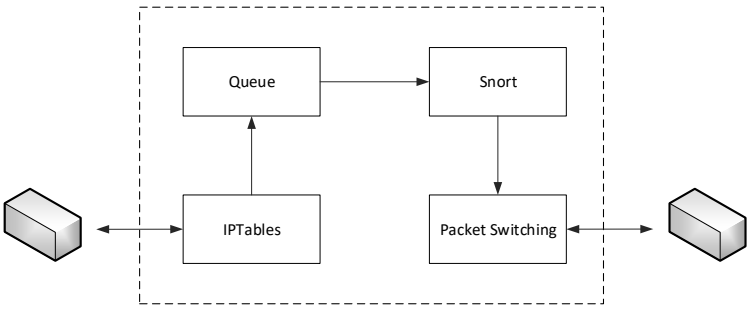

Figure 3.1: Snort's inline mode [18]

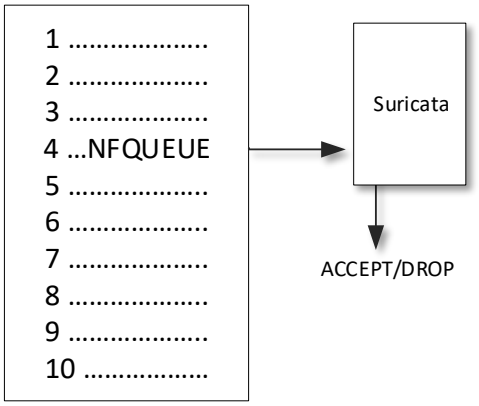

Figure 3.2: Suricata's iptables and NFQ Mode: accept [47]

work the host organization should consider a proof-of-concept based on NFLOG which allows nftables to send copies of the packets instead of queuing them. This can set a pre-implementation phase for the whole project or either be used as a Development, Testing, Acceptance and Production (DTAP) approach when deploying new rules to the systems. Looking for a more custom approach for the host organization, we foreseen an implementation that could be based on the mixed mode suggested by Giuseppe Longo [50] [49] where an nftables' rule would log traffic with NFLOG to a listening IDS instance/configuration and another would queue traffic with NFQUEUE to a listening IPS instance/configuration. As an example, such approach allows a scenario where web server accesses can be just logged (never to be inadvertently blocked by the IPS, just checked by the IDS) and remaining traffic would be queued to be fully inspected.

3.2 Rules

An IDSs rule is a method to perform detection. It is an expression processed to look for matches in data or properties, detecting a vulnerability and reacting accordingly by per-

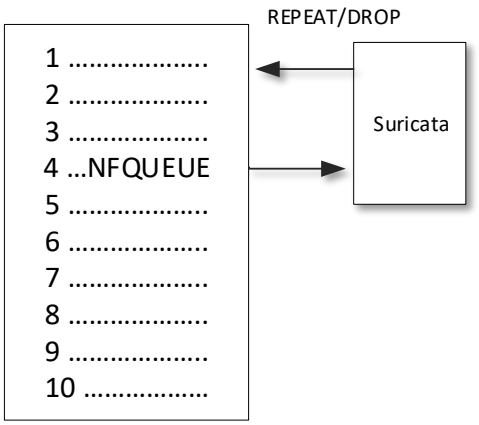

Figure 3.3: Suricata's iptables and NFQ Mode: repeat [47]

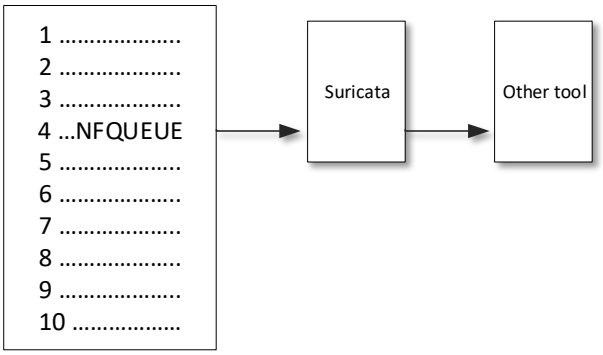

Figure 3.4: Suricata's iptables and NFQ Mode: route [47]

forming some action as described in it. Snort advises to look in to catching vulnerabilities rather than specific exploits, since this makes the rule less vulnerable to evasion if the exploit is slightly changed or mangled [43]. Snort's rule base allows its language to combine signature inspection methods, protocol inspection methods and anomaly-based inspection methods. Suricata started with the same approach as Snort's, later developing its own language. Pre-existing rulesets can be used or one can customize rules as needed for both Snort and Suricata. Both IDSs are widely compatible with the rules developed for such systems. In Table 3.1 we can find a list of the main rulesets provided and recommended by Snort and Suricata's developers and maintainers. Because of the variety of rulesets and their sources, also because of their maintenance (add, remove, amend) and the associated tasks (download, combine, install, update), it is advisable to use a rule management application. In Table 3.2 we find a list of suggested rule managers, both compatible with either Snort or Suricata. The developers provide free access to the rulesets, but they also make available a privileged and faster access to the rules, provided through a subscription, with

Table 3.1: Rulesets

Ruleset
Shared Object Rules [39]
Snort Rules Snapshot (aka VRT aka Talos Rules) [41]
Community Rules [6]
Talos IP Blacklist [51]
Emerging Threats Open [10]
Emerging Threats Pro [11]

Table 3.2: Rule Management programs

Program
PulledPork [36][35]
Oinkmaster [28]

other extra features added. For Snort they claim a 30 days faster access for registered users, as for Emerging Threats they claim daily updates for subscriptions.

Listing 3.1: Example of Snort loaded rules' summary with emerging.rules snortrules community ip-blacklist managed by Pulledpork

```
29898 Snort rules columnsread
        28709 detection rules
        150 decoder rules
        268 preprocessor rules
29127 Option Chains linked into 3212 Chain Headers
0 Dynamic rules
```

Listing 3.2: Excerpt of PulledPork unloading snortrules community ip-blacklist

```
Rule Stats...
        New:-------9
        Deleted:---33188
        Enabled Rules:----18375
        Dropped Rules:----0
        Disabled Rules:---6366
        Total Rules:------24741
```

Listing 3.3: Example of Snort loaded rules' summary with emerging.rules managed by Pulledpork

```
18375 Snort rules columnsread
    18375 detection rules
    0 decoder rules
    0 preprocessor rules
18375 Option Chains linked into 2670 Chain Headers
0 Dynamic rules
```

Table 3.3: Rulesets used in this work

Community/Open	Registered	IDS
ip-blacklist		Snort2
community-rules.tar.gz	snortrules-snapshot.tar.gz	Snort2
snort3-community-rules.tar.gz	snortrules-snapshot-3000.tar.gz	Snort3
emerging.rules.tar.gz		Snort2/Suricata

Listing 3.4: Example of Suricata loaded rules' summary with emerging.rules managed by Oinkmaster

```
dd/mm/yyyy -- hh:mm:ss - <Info> - 38 rule files processed.
   ↪ 12529 rules successfully loaded, 0 rules failed
dd/mm/yyyy -- hh:mm:ss - <Info> - 12534 signatures
   ↪ processed. 1158 are IP-only rules, 5309 are
   ↪ inspecting packet payload, 7656 inspect application
   ↪ layer, 0 are decoder event only
```

3.3 Snort2

It is largely pointed that Snort is single-threaded, but this is truth for a single running instance, others tasks can be multi-threaded such as reloads, IP reputation switching [23] or the outputs' processing and output handling (Barnyard2 and/or SQL) can be assigned to other cores. Several instances can run concurrently on different cores using different rulesets, analyzing the same stream with different rulesets, or balancing the instances to analyze different streams with the same rulesets. Snort with PF_RING along with socket clustering allows to distribute packets across multiple processes [42]. Figure 3.5 depicts Snort's pipeline which has been immutable since early versions [27]. This architecture's layout is extended in Figure 3.6 to allow a broader interpretation of the application's flow. The Data Acquisition library (DAQ) acquires the packet which is delivered to the packet decoder. Each pre-processor is engaged according to its features which later passes its results to the detection engine. This is where the available rules are checked against the packets being processed, resulting in a type of output parsed by output plugins. At this stage we should have a reaction from the processing, but, it is when the output facilities are triggered, allowing to either alert or log the output. Depending on the interaction intended from the system its user can have the alerts directed to an interface, allowing a real-time interaction, or either log it along or after the alert. In this case, the log facility can address this task in several ways, by using plain text or binary log files, writing to a Database (DB) or send out through syslog. For an optimized performance it is advised to get the IDSs log facility to always output to binary files (Unified2 File Format), as this reduces the translation efforts and the delays in packets processing. The task of translating

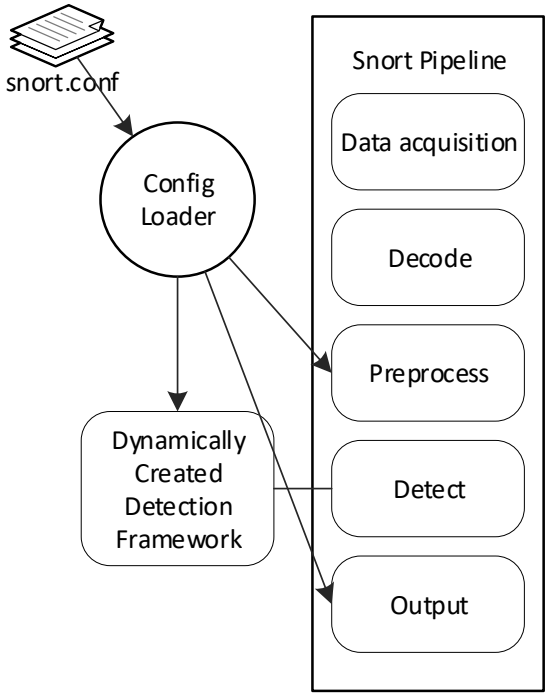

Figure 3.5: Snort's pipeline [27]

the binary files, either to write them to a DB or to read them, should be separated and left to a different asset of which Barnyard2 [4] is a good example. As depicted in Figure 3.7, the output can be sent out through syslog to a remote management or log collector, which permits to add another layer of inspection, for example using an event correlation tool or a Security Information and Event Management (SIEM), something addressed below.

3.4 Suricata

Suricata is developed and sustained as Open Source under a GPLv2 licence. Its motivation was to address some gaps in existent systems, and to explore additional capabilities unexplored when the project began. Is was built to address high performance with parallelized processing, explore other hardware opportunities (GPU acceleration or hardware sensors) and also to implement new approaches on detection, decompression or matching. Therefore it is expected to perform faster out-of-the-box. Suricata also includes standard input and output formats to allow swift, flexible and effortless integration with other existing tools such as dashboards, log collectors or SIEMs [45]. Figure 3.8 shows that Suricata's base pipeline (e.g. PCAP device runmode) is pretty much the same as for Snort. Nonetheless, in Figure 3.9, we can see how the threading is managed through four thread-modules in its default run mode. The packet acquisition module is responsible for reading the

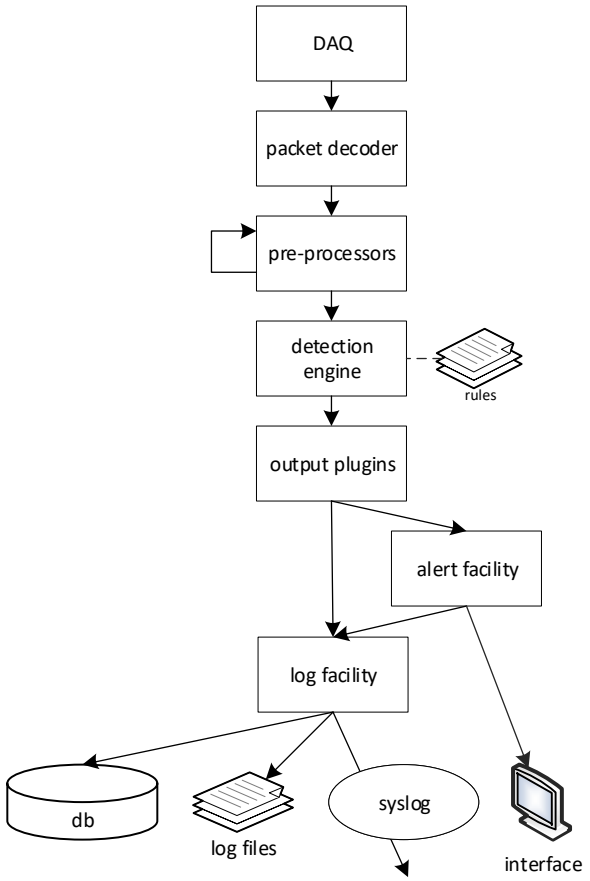

Figure 3.6: Snort's facilities

packets from the network. The second module addresses the packet decoding, but also the stream application layer which includes three tasks: stream-tracking, stream-assembly and application layer inspection. Detection threads are used to compare signatures and can operate simultaneously. Lastly, the outputs module, processes all alerts and events. Figure 3.10 shows how in *pfring* runmode each flow follows its own fixed route. However it should be noted that the focus is in the detection task, the more demanding task, that checks packets against thousands of signatures [46]. Additionally Suricata includes other performance customization options, such as CPU affinity, which allows to set fixed cores for every thread, here represented in Figure 3.11.

3.5 Snort3

The new version of Snort seems to have been renamed or made known by several names. The Snort++ (codename) project, aiming to develop Snort version 3, is sometimes called just by Snort Alpha. Snort version 3 has been in development at least since 2005 when its

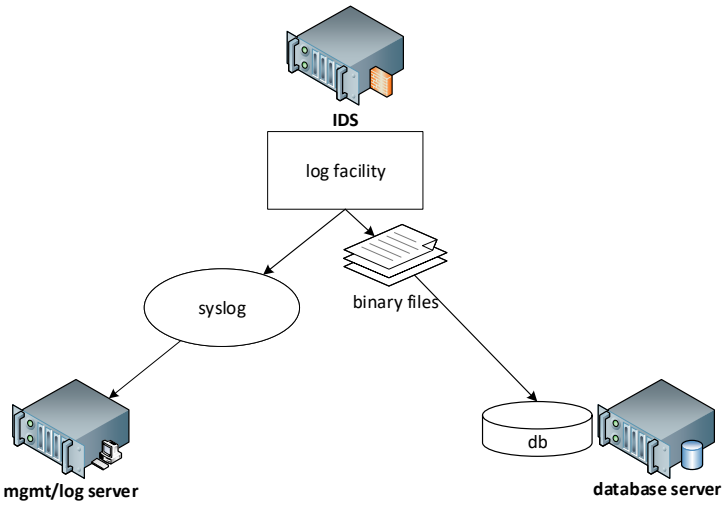

Figure 3.7: Snort's binary and syslog output

author called the project SnortSP (Snort Security Platform), aiming to rethink its concepts and architecture. But around 2014 is when the Snort development team announces a new Alpha version to be tested in the wild [21]. Its concept was presented by Martin Roesch in a 2008 presentation [27] where he shared its architecture's conception, shown in Figure 3.12. Besides a completely new code base, its architecture aims to be multi-threaded and accelerated, with engines running continuously with no need for reloads and multi-core parallelization taking advantage of modern hardware. The figure emphasizes the modularity of Snort3, showing how each abstraction can be accelerated and tunned not only individually but also while communicating and sending tasks to others. As for rules, Snort3 also uses a new rule's language. Despite being very similar, it is not the same. Therefore Snort is providing new rulesets on their website, both community and Talos' (Cf. Table 3.3). At the moment of our writing PulledPork is not yet compatible with Snort 3, but a milestone has been shared by the project's maintainers.

Though we wanted to add this novelty to our work, Snort advertises that this product is only an (yet another) Alpha version and therefore it is not ready for production and should not be used for such purpose.

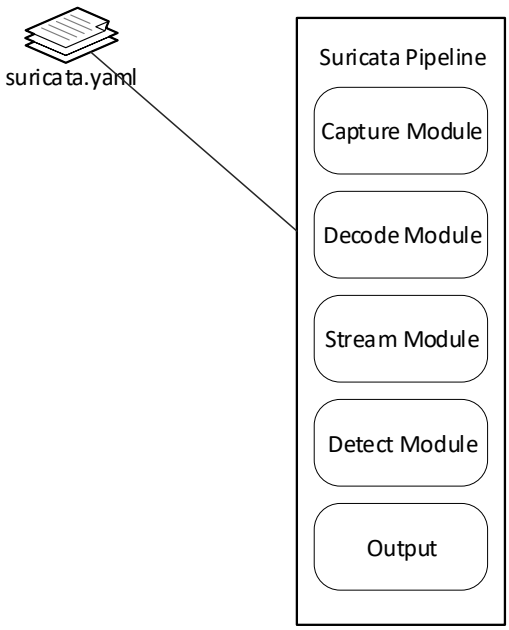

Figure 3.8: Suricata's pipeline (based on PCAP device runmode) [46]

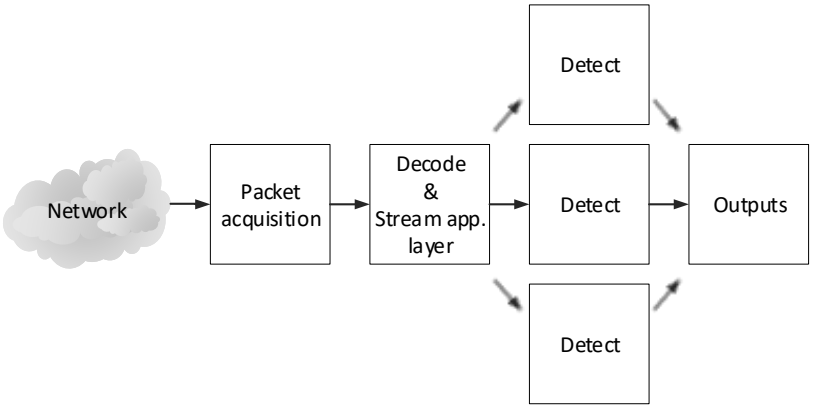

Figure 3.9: Suricata's default runmode

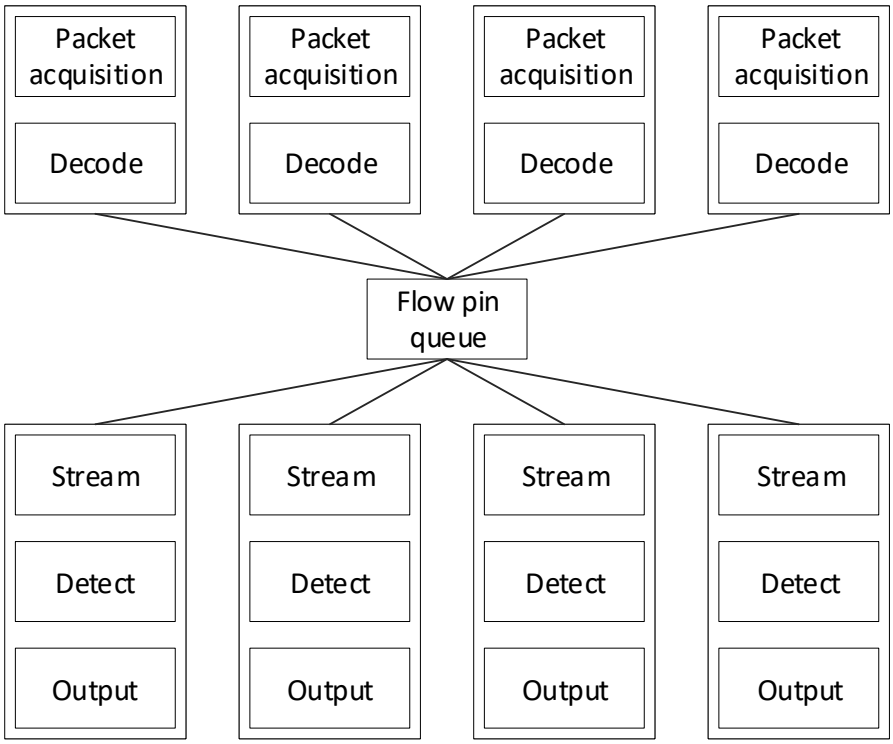

Figure 3.10: Suricata's pfring runmode

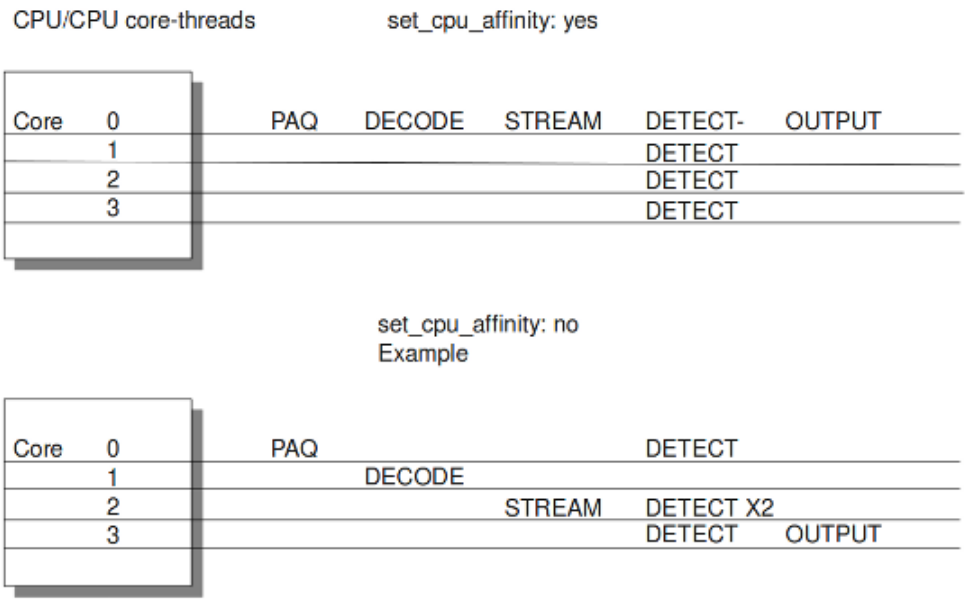

Figure 3.11: Suricata's CPU affinity schema

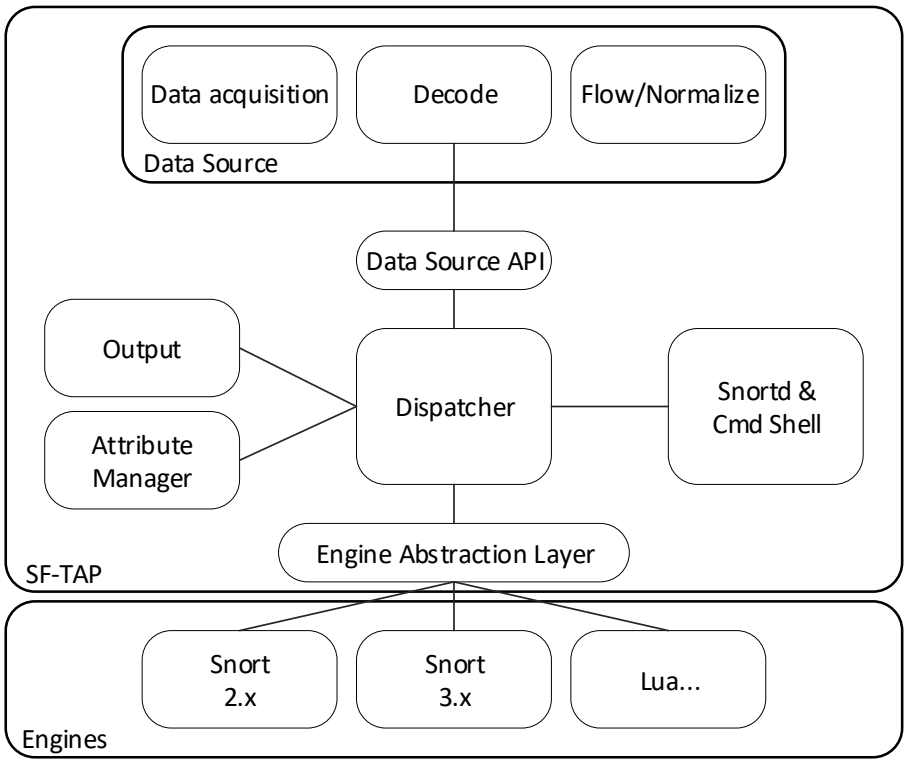

Figure 3.12: Snort3's pipeline [27]

This page intentionally left blank.

Chapter 4

Design

4.1 IDS general approach

A basic requirement for an IDS implementation is the access of the sensor to all traffic in the network to be protected/surveilled. Figure 4.1 depicts an alternative for its satisfaction using a Test Access Point (TAP). TAPs pass copy of all the traffic at the network segment to the sensor. While this approach seems to be simple to implement its downside include additional hardware and costs, or to create a disruption to introduce the device. Another alternative is to use a port mirror, as shown in Figure 4.2. This approach, normally available in standard switches (with capacity limits per model though) is scalable, flexible, configurable and in general less expensive. On the negative side, both approaches present some limitations. Switched Port Analyzer (SPAN) can result on drastic bottlenecks at the mirror port and hides port errors from the sensor. TAP would include hardware or port errors and packets wouldn't be dropped.
Though these details should not be forgotten for the host organization's implementation and real scenario tests, they will not be discussed further in this document since the available and most suitable option is based on port mirroring using Switched Port Analyzer (SPAN) sessions.

4.2 Lab approach

Kernel-Based Virtual Machine (KVM) allows a swift creation of virtual environments and a prompt way to scale solutions. Because of acquaintance and access to such resource we have selected this environment to kick off our laboratory so we could run initial simulations and experiments, by this creating drafts for later stages. As depicted in Figure 4.3 the project used a KVM host with two Network Interface Controllers (NICs) allowing to simulate an internal and an external network, or, if we want, an internal and a Demilitarized Zone (DMZ), since ahead we had a router with a built-in firewall. An iptables based firewall was implemented between those NICs, and (external) adapter (br0 in Figure 4.3)

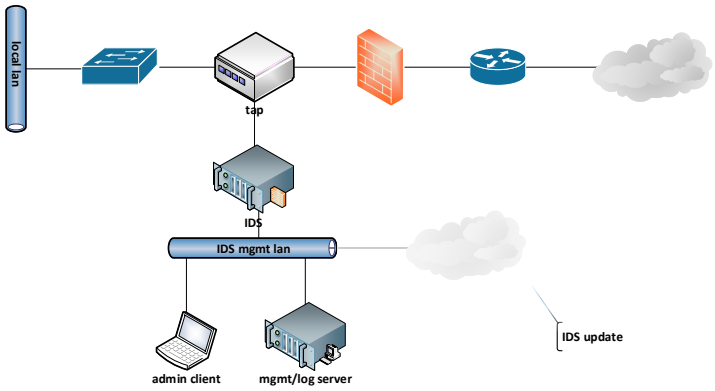

Figure 4.1: HLD with TAP

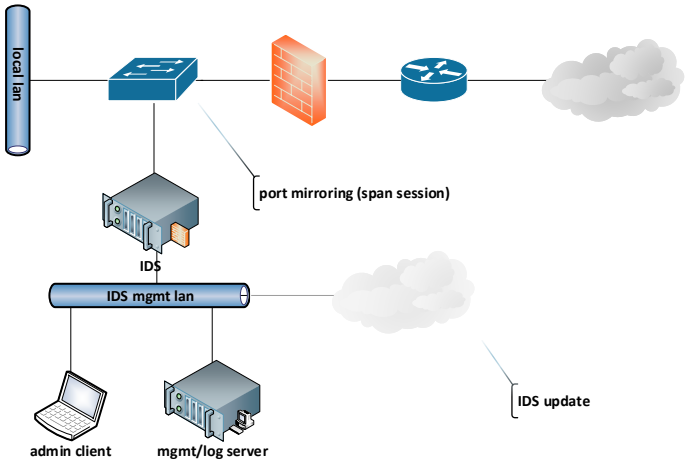

Figure 4.2: HLD with port mirroring

was set to act as a hub. This was achieved by setting the Linux bridge to forget immediately any MAC address seen, and to forward with no delay. Packets received on br0 were now seen from any VM with an active interface using this network adapter (with no address). With this method, instances of IDSs where receiving packets from the listening interface, acting as sensors, and having another network interface in the local (internal) LAN for management but also for product or rules' updates etc. At a later stage, SIEMs were set on the internal LAN, receiving syslog messages from the IDSs through the local network.

4.3 IPS in production

Despite this work has been focused in comparing the IDSs performance, the objective of the host organization was to assess its capacities to set it as IPS in inline mode. Nonetheless, its implementation has been brainstormed. It is possible to achieve High Availability

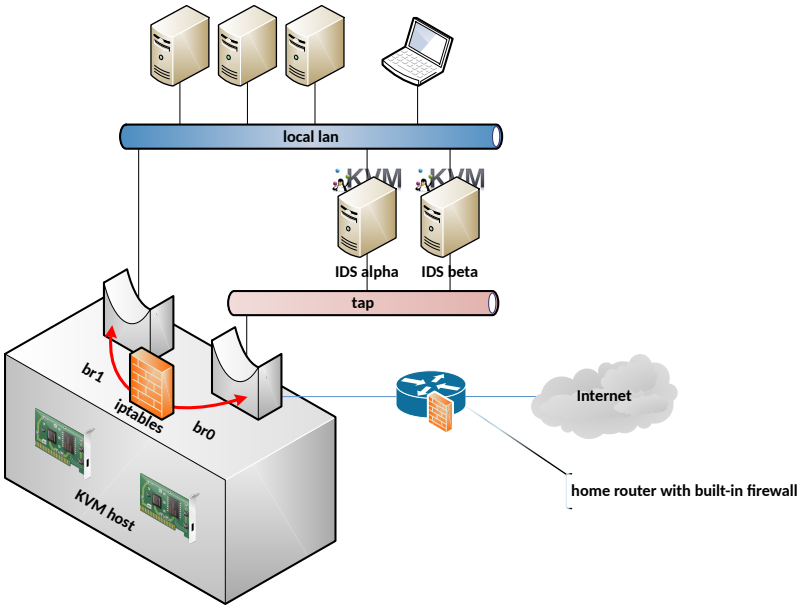

Figure 4.3: HLD for KVM laboratory.

(HA) for iptables using keepalived to provide load balancing and conntrackd to track netfilter monitored connections. But without a clear path to achieve the same robust solution for one of the IDSs in our testbed, suggestion is to pursue the approach used in Cisco Adaptive Security Appliance (ASA) products with Sourcefire (later FirePOWER) module/service. Here each hardware firewall in a failover pair, implements the inspection module by software (Snort2) in a Linux based image, using a dedicated module. The module on the primary device does not synchronize with the module on the secondary device. The caveats, separate/duplicate configuration management and temporary abnormal behavior resulting from the absence of state sharing between the two replicas. To the extent of our knowledge, no solution similar to Cisco's Management Center that provides centralized configuration exists for both Suricata and Snort. Other Sourcefire/Cisco products provide support for HA but not Snort per se. We also couldn't find it for Suricata. However if the implementation is supported by a virtualization environment, HA capabilities can be provided from the hosting perspective, as long as packets' capture techniques employed are supported over virtualization and do not reduce its performance (as the ones presented by ntop [1]). Taking Figure 4.4 as an example, we can achieve this model with iptables (with keepalived and conntrackd) but with the above-mentioned caveats, the IPS would be implemented per firewall server.

4.4 PCAP Offline mode

IDSs were compared using several testbeds, from regular home desktop hardware, to cloud based virtualization options and later to bare-metal. These testbeds allowed to

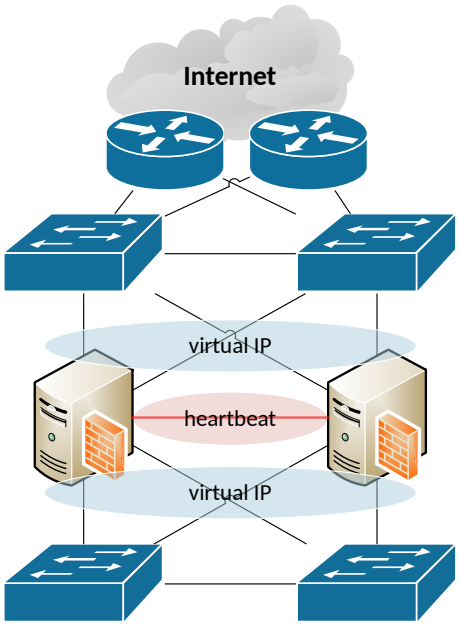

Figure 4.4: HLD for iptables with HA.

Table 4.1: PCAP Sources

Sources
http://tcpreplay.appneta.com/wiki/captures.html
http://contagiodump.blogspot.pt/
http://maccdc.org/

create a base for comparison using a specific set of Packet Capture (PCAP) files, running all of them on these environments and changing the available resources in each test. The PCAP files selected for the comparisons were obtained from several different sources, to avoid moving private packet captures to outside the host organization. Those sources are listed in Table 4.1. Those captures have been bundled in ”Short PCAPs” and ”Long PCAPs”, allowing to have a smaller set for quick tests and a larger set for longer tests. Tables 4.2 and 4.3 show the list of the mentioned PCAPs and their sizes. Their traffic varies from real network traffic on a busy private network’s access to Internet [52], to attack sessions and exploit attempts [7] or cybersecurity competition’s sessions [26] in the case of the larger files.

4.5 IDS in pre-production

To take the implementation to pre-production stage, an approach similar to the one in Figure 4.2, using SPAN sessions to deliver traffic to our bare-metal environment was used. A representative number of LANs have been selected to be used in our Proof of Concept (PoC), allowing us to test both public access traffic and purely internal user’s

Table 4.2: Short PCAPs (statistics collected with tcpstat)

Description	Packets	Size
99.1.23.71.pcap	5544	2.5M
analysis.pcap	1835	909K
BIN_9002_D4ED654BCDA42576FDDFE03361608CAA_2013-01-30.pcap	6661	4.0M
BIN_LoadMoney_MailRu_dl_4e801b46068b31b82dac65885a58ed9e_2013-04.pcap	43147	30M
BIN_Tbot_23AAB9C1C462F3FDFDDD98181E963230_2012-12.pcap	5004	3.3M
BIN_Tbot_2E1814CCCF0C3BB2CC32E0A0671C0891_2012-12.pcap	6908	4.1M
BIN_Tbot_5375FB5E867680FFB8E72D29DB9ABBD5_2012-12.pcap	8000	5.2M
BIN_Tbot_A0552D1BC1A4897141CFA56F75C04857_2012-12.pcap	4787	4.0M
BIN_Tbot_FC7C3E087789824F34A9309DA2388CE5_2012-12.pcap	13050	7.5M
pcap_4B31A4C3A633A0ADB9DBB8A5125DDA85.pcap	138	31K
pcap_59A14B490FE4BA650E31B67117302239.pcap	68	9.0K
pcap_9B41475A88D12183048A465FFD32EBF9.pcap	982	323K
pytbull_ALL.pcap	286727	44M
pytbull_DOS.pcap	3312	2.4M
pytbull_replay.pcap	1452	2.3M
XTremeRAT_DAEBFDED736903D234214ED4821EAF99_2013-04-13.pcap	2729	3.6M

Table 4.3: Long PCAPs (statistics collected with tcpstat)

Description	Packets	Size	μ	σ
bigFlows.pcap	791615	352M	434.98B	575.75B
maccdc2011_00010_20110312194033.pcap	10000000	3.7G	360.83B	286.60B
maccdc2011_00011_20110312201409.pcap	5000000	3.0G	598.96B	501.71B
maccdc2011_00012_20110312202052.pcap	5000000	3.2G	654.70B	532.54B
maccdc2011_00013_20110312202724.pcap	10000000	4.2G	413.06B	590.26B
maccdc2011_00014_20110312233311.pcap	4465786	2.5G	565.77B	646.96B
purplehaze.pcap	324711	231M	714.08B	690.10B

μ Average
σ Standard Deviation

traffic. The bare-metal sensor had 2 listening interfaces, with 10Gbps of port speed, with up to 4 interfaces available, plus 1 dedicated to management and syslog messages (the hardware was a Dell R430 server with 2+1+mgmt interfaces at 1Gbps plus 2xNICs with 2x10Gbps interfaces each). One of the listening interfaces was receiving traffic from the external interface on a front-end firewall. The traffic includes both the published services of the host organization and the Internet traffic generated by the internal network and its users. For this one, the IDS was set to protect two times /16 and one /21 networks (*$HOME_NET*). It is worth saying that the /21 network was public IP addressing, where more than one hundred websites are published. The second interface was receiving traffic from a core switch with mirrored traffic from more than 100 VLANs. These included a /16 and a /23, both set to be protected as *$HOME_NET* in our PoC. For this second one a graph with one month's network statistics is shown in Figure 4.5.

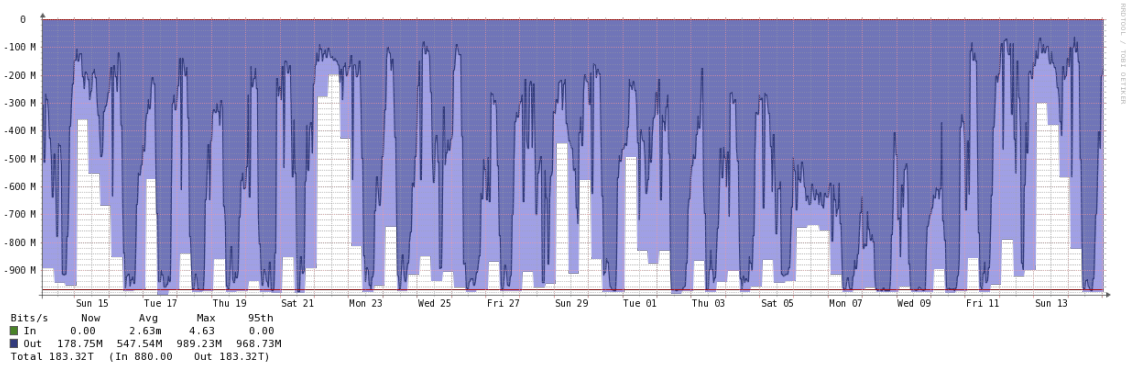

Figure 4.5: Internal network, one month stats.

Chapter 5

Implementation

5.1 Testbeds for PCAP Offline mode

To achieve a broader view with several implementation scenarios (e.g. physical versus virtual), and to circumvent the hardware availability limitations, which would have given us the opportunity to test both the IDSs side by side, we went for an approach based on packet captures. To achieve credible and comparable values, several environments were tested ranging from KVM in common desktop, to Physical dedicated server. Table 5.1 depicts the 4 distinct testbeds as well as the short designation used throughout the text. All the environments used Debian 9.Table 5.2 presents the hardware resources available on each. The first environment was built in a home desktop using KVM and was depicted in Figure 4.3. This desktop had an Intel® Core™ i7-4770K CPU @ 3.50GHz permitting to allocate up to 8 logical CPUs. As for VMware, the product available was based on Intel® Xeon® CPU E5-2683 v3 @ 2.00GHz, allowing allocation of up to 56 virtual logical processors. Similarly, the VPS product, was based on Intel® Xeon® E5-2687W v4 @ 3.00GHz cores with options in 2, 4, 8, 16 and 32 virtual logical processors. Finally, our physical environment, was seating on a Dell R430 with Intel® Xeon® CPU E5-2623 v3 @ 3.00GHz with 16 logical CPUs.

5.2 Testbeds for IDS

As a testbed to simulate, troubleshoot and create a base setup for IDSs, the testbed presented above was configured with dynamic resources allocation to allow a swift adaptation along the research. It was possible to, firstly, create a base network with two abstractions, an outer and an inner network zones. These zones were separated by an iptables based firewall and set with the outer interface bridged to allow all traffic hitting this firewall to reach the IDSs listening interfaces. This was achieved using brctl's options *setageing* and *setfd* to 0 so that MAC addresses were not saved and packages are forwarded with no delay [5], allowing for a hub alike behavior at the host level. This context allowed us to

Table 5.1: Testbeds

Ref.	Description
KVM	Home desktop
VMW	Virtualization cluster
VPS	Cloud service
MET	Pre-production bare-metal hardware

Table 5.2: Testbeds resources' details

Ref.	CPU	CPUs allocated	RAM allocated	SSD
KVM	Intel Core i7-4770K 3.50GHz	1-8	8GB	Yes
VMW	Intel Xeon E5-2683 v3 2.00GHz	4,8,16,32	8GB	Yes
VPS	Intel Xeon E5-2687W v4 3.00GHz	4,8,16,32	15,30,60,120GB	Yes
MET	Intel Xeon E5-2623 v3 3.00GHz	1-8,16	32GB	No

safely generate traffic at the external network while keeping the internal network's assets secured. Issuing new VMs, duplicating, snapshotting or changing resources at the KVM environment was easy and instantaneous. IDSs were deployed with two interfaces, with one set in the internal network for management and another in the "tap network", as seen in Figure 4.3.

As found in the appendices, we've brainstormed many options across the System Under Test (SUT). We've tested rule's managers, start up scripting and various log and alert options as described in each IDSs section. It was possible for us to build quick setup guides (for Debian 9), that can be publicly shared and also used by the host organization's systems administrators to install and maintain.

5.3 IDS

5.3.1 Snort

Our implementation of Snort is documented in appendix A. It's based on *Snort 2.9.9.x on Ubuntu 14 and 16* by Noah Dietrich from Snort's Documents page [40], a guide to setup Snort on Ubuntu 14 and 16. This appendix includes basic applications to help troubleshooting (such as tcpdump), the setup of the listening interface, review of dependencies to build Snort from source and the creation of Snort's directories. Though we've started with Snort's version 2.9.9.0 soon we've changed to 2.9.11.1, released in October 2017. Table 5.3 shows Snort's directories and, in code snippet 5.1, we can check Snort's folder structure.

Listing 5.1: Snort's folder structure

```
/etc/snort/
```

Table 5.3: Directories' summary

Purpose	Path
Snort binary file:	*/usr/local/bin/snort*
Snort configuration file:	*/etc/snort/snort.conf*
Snort log data directory:	*/var/log/snort*
Snort rules directories:	*/etc/snort/rules*
	/etc/snort/so_rules
	/etc/snort/preproc_rules
	/usr/local/lib/snort_dynamicrules
Snort IP list directories:	*/etc/snort/rules/iplists*
Snort dynamic preprocessors:	*/usr/local/lib/snort_dynamicpreprocessor/*

```
|-- attribute_table.dtd
|-- classification.config
|-- file_magic.conf
|-- gen-msg.map
|-- preproc_rules
|-- reference.config
|-- rules
|       |-- iplists
|       |       |-- black_list.rules
|       |       |-- white_list.rules
|       |-- columnslocal.rules
|-- sid-msg.map
|-- snort.conf
|-- so_rules
|-- threshold.conf
|-- unicode.map
```

We opted to use PulledPork to manage the rules, but the fact is that, for the host organization, a controlled approach should be more suitable, allowing to dissect and analyse new rules as they are provided. It is worth mentioning that Snort needs to be restarted once rules are updated or configuration changes are submitted. Recent versions allow for a reload submission depending on the type of changes made and with caveats regarding the ongoing sessions. Moreover this option is available if enabled during compiling. As for logging, we've tested Barnyard2, providing binary option for outputs, to allow reduction of resource's consumption. Nonetheless we found advisable to move this database to an external system, allowing dedicated resource's allocation and the possibility to maintain this as a separate and modular environment. This abstraction also allows to migrate this piece of the system to another type of environment, for example to a log collector or event manager, by either creating it for such purpose or aggregate it to an existing one. For both, PulledPork and Barnyard2, startup scripts have been tested and are provided in

the appendices, along with some maintenance and troubleshooting commands for Barnyard2's database.

During its implementation, in the bare-metal environment, techniques as Large Receive Offload (LRO), Large Send Offload (LSO) or Generic Receive Offload (GRO) have been considered; nonetheless those are documented, for example by Snort, as to be related with possible packet loss during reassembly [33]. There is a possible mismatch of MTU between packets reassembled by the NIC and the truncate value used by Snort. As such, we have opted to remove such options from the NIC to avoid misleading behaviors implemented by the network adapter's vendor. Also, as highlighted in the same document, promiscuous mode (Listing 5.2) has been enabled to avoid the NIC to stop traffic from reaching the sensor. Listings 5.3 and 5.4 show the configuration of the NIC using ethtool.

Listing 5.2: ifconfig promiscuous mode examples

```
#ifconfig enable promiscuous mode example with eth0
ifconfig eth0 promisc
#ifconfig disable promiscuous mode example with eth0
ifconfig eth0 -promisc
```

Listing 5.3: ethtool set features example

```
#ethtool set features example with eth0
ethtool -K eth0 tx off rx off tso off gso off gro off
```

Listing 5.4: ethtool show features example

```
#ethtool show features example with eth0
ethtool -k eth0
Features columnsfor eth0:
rx-checksumming: off
tx-checksumming: off
        tx-checksum-ipv4: off
        tx-checksum-ip-generic: off [fixed]
        tx-checksum-ipv6: off
        tx-checksum-fcoe-crc: off [fixed]
        tx-checksum-sctp: off [fixed]
scatter-gather: on
        tx-scatter-gather: on
        tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: off
        tx-tcp-segmentation: off
        tx-tcp-ecn-segmentation: off
        tx-tcp-mangleid-segmentation: off
        tx-tcp6-segmentation: off
udp-fragmentation-offload: off [fixed]
generic-segmentation-offload: off
generic-receive-offload: off
```

```
large-receive-offload: off
rx-vlan-offload: on [fixed]
tx-vlan-offload: on
ntuple-filters: off [fixed]
receive-hashing: on
highdma: on [fixed]
rx-vlan-filter: on
vlan-challenged: off [fixed]
tx-lockless: off [fixed]
netns-columnslocal: off [fixed]
tx-gso-robust: off [fixed]
tx-fcoe-segmentation: off [fixed]
tx-gre-segmentation: on
tx-gre-csum-segmentation: on
tx-ipxip4-segmentation: on
tx-ipxip6-segmentation: off [fixed]
tx-udp_tnl-segmentation: on
tx-udp_tnl-csum-segmentation: on
tx-gso-partial: on
tx-sctp-segmentation: off [fixed]
fcoe-mtu: off [fixed]
tx-nocache-copy: off
loopback: off
rx-fcs: off [fixed]
rx-all: off [fixed]
tx-vlan-stag-hw-insert: off [fixed]
rx-vlan-stag-hw-parse: off [fixed]
rx-vlan-stag-filter: off [fixed]
l2-fwd-offload: off [fixed]
busy-poll: off [fixed]
hw-tc-offload: off [fixed]
```

Experiences with Snort created the opportunity to interact with the support team, for submission of false positives and report of typos on the web site. Responses were very prompt and satisfactory specially in the IRC channel, which we considered to be very active when compared with Suricata's, PulledPork's, Barnyard2's or Emerging-Threats'.

5.3.2 Suricata

Appendix C presents basic Suricata's setup steps, based on Suricata's Wiki for Debian Installation, and Basic Setup, both available at Suricata's Wiki [47]. As done for Snort, we have included troubleshooting applications and steps related to the OS; dependencies have been reviewed and Suricata was built from source.

With Suricata we have used Oinkmaster to manage rules and we opted to set Emerging-Threats as a base feed for Suricata's initial setup. We found both with a similar approach

to configure, start, or run with options. Nonetheless Suricata seemed to be far more simple to install and start, with just a few steps. Though versions 3.2.4 and 3.2.5 were released during the period of our experiments, we've started with version 4.0, available since July 2017. We've also seen 4.0.1 and 4.0.3 being released and we have finished with 4.0.4 since February 2018 (4.1 beta 1 ready for testing since March 2018). This seems to demonstrate how active and alive this project is. Also, in late 2017 Suricata-Update (rules' update manager) has been announced; nonetheless we hadn't the chance to include it in our testing. We found Suricata easy to setup, very customizable with a modular approach and compatible proof, allowing it to be implemented with other external systems. A lot of documentation is available and its manuals and guides are quick and short to read, again, what seems to be a very modular approach.

NIC, drivers and kernel related techniques were set using the same approach of Snort. Suricata also documents such approaches to be disabled or else merged packets won't be correctly identified and TCP state tracking can be broke. Yet, Suricata documentation states that checksum offloading can be left enabled for both AF_PACKET and PF_RING. Other considerations can be found in Suricata's User Guide under the Packet Capture section [46].

5.3.3 Snort3

Though we experimented much less with Snort3 we have included its setup in Appendix E. Snort3 shares similar dependencies, yet some are different and require independent setup. There is a requirement for a legacy C-library which we found odd and worth mentioning. It includes the Hyperscan library for high-performance multiple regex and pattern matching which we also found to be used by Suricata. Nonetheless Hyperscan needed to be setup with other libraries being provided and referenced. Also requires FlatBuffers, a cross platform serialization library, and a particular version of DAQ which is not the same as for Snort2. Remaining setup and usage is quiet similar, but again, not fit for production and therefore we haven't spent much time with it beyond some lab and PCAP runs.

5.4 SIEM

The selected SIEM for our testing was Open Source Security Information Management (OSSIM), an Open Source version of AlienVault's SIEM. Its features include Security Information Management, Security Event Management, Asset Discovery and Asset Management, Log Management, Network and Asset Monitoring, IDS, HIDS, Vulnerability Assessment, Threat Detection, Behavioral Monitoring, Reporting, Incident Response and others. OSSIM includes tools such as Nagios or OpenVAS or even Suricata, with plugins compatible with log collection for dozens of vendors, plus AlienVault's Open Threat

Exchange (OTX) a community driven threat intelligence service leveraging information from many different sources, to provide both up-to-date information and fresh event correlation.
We've started to experiment with OSSIMs live demo at their website where we soon found a clean interface that could help us dealing with our main requirement, to read the IDSs alerts. We've later installed version 5.4, available since June 2017, but ended this work with version 5.5 after quiet a few fixes and updates. Such number of updates made clear that this project is maintained and kept up-to-date. The product is simple to install but yet we found some caveats. Along with resource greed, OSSIMs image seems to be packed with limited drivers, this led us to some installation delays, but nothing relevant. We found 4 CPUs and 8GB of RAM to be a decent value for OSSIM to work normally with 1 or 2 simultaneous syslog feeds. As for disk space, the value has not been assessed since it is dependent of the amount of logs to be collected and stored, retention periods etc. It is worth mentioning that such tools across vendors are normally setting a base requirement around the order of 500GB to 2TB of disk space.
OSSIM provides several options for integration with external feeds. Yet our experiments were based on syslog only as we just wanted to focus on testing the presentation and interaction rather than its performance. Other possibilities could be assessed though, such as built-in IDS (Suricata) and binary feeds rather than syslog. As such, also other plugins could have been tested such as Nagios' integration or its built-in features and OpenVAS for vulnerability scanning.
It is worth mentioning that, in our lab environment, we had the chance to set Snort and Suricata side-by-side and so did we with OSSIM, setting one instance per IDS, with both listening to the same traffic. This allowed us to get separate looks and feelings for each integration.

5.5 Other tested options

5.5.1 Pytbull

While testing in lab, we have tried to use Pytbull, a python based IDS/IPS testing framework. Unfortunately we soon gave up, as we found it to be more HIDS oriented and apparently ready to work only if installed in the IDS sensor rather than working on the network as client and server. In fact its documentation seems to lead to this possibility, yet even pointing FTP at the sensor for alert's feed, we couldn't make it to work in the time we had reserve for this attempt. Its setup can be found in the appendices in D. Pytbull's latest version is 2.1 which was released in March 2016 with no later updates. As already said, our work was more focused on performance comparison rather than rules and alerts, because those can be customized and used in both platforms and therefore we abandoned this approach.

This page intentionally left blank.

Chapter 6

Results

6.1 PCAP Offline mode

From the PCAP files selected for this work (introduced in Section 4.4), we ran a total of 1541 tests (described in Table 6.1). From that list, 469 we considered to be long enough to provide usable results, since the other 1072 were too short, leading to runs of just a few seconds. These short tests resulted in incorrect and incomplete reads being parsed by the scripts we've used. Yet, those have been useful when testing the scripts or testing the first runs in each environment, ensuring that we could leave the long ones running unattended. With these tests, we collected information from the console output, both IDSs filtered statistics, but also statistics from the system using pidstat from sysstat (a system performance tool for the Linux operating system). We have collected details on memory, processing, faults, times and packets. Both, the batch and the parsing scripting, were based on White's [12] scripts. The resulting parsed data was later plotted, where relevant, as in appendix F. Out of these plots, we are reproducing some in this text to support our analysis. The metrics used were measured in Packets per Second (PPS) (packets processed by the IDS), number of allocated CPUs (Central Processing Unit) and Resident Set Size (RSS) (memory occupied by a process in kB).
Looking at Suricata's results it seems clear that files too small lead to possible erroneous values because of statistics parsing. As depicted in Figures 6.1 and 6.2, both these captures seem to be so short as to be processed in just a few seconds, before the next statistics are retrieved.

Using (large) 10M packets captures as reference (6.3 and 6.4), we saw both, the KVM and the bare-metal environments, consistently outperforming VPS and VMware environments. This was seen for Suricata and also for Snort2. We would expect this behavior while on NICs acquisitions, but not reading from a file. We've seen some documentation debating on techniques to reduce such differences for virtual environments such as VMware, but those are related to NICs drivers overhead and the results are not applicable to differences to reading and parsing from PCAP files.

Table 6.1: PCAP tests

Ref.	Batch name	Tests	CPU sets	PCAP sets
KVM	kvmlong	112	8	7
KVM	kvmshort	256	8	16
KVM	kvmalphalong	56	8	7
KVM	kvmalphashort	128	8	16
VMware	vmwlong	56	4	7
VMware	vmwshort	128	4	16
VPS	vpslong	56	4	7
VPS	vpsshort	128	4	16
bare-metal	metlong	126	9	7
bare-metal	metshort	288	9	16
bare-metal	metalphalong	63	9	7
bare-metal	metalphashort	144	9	16

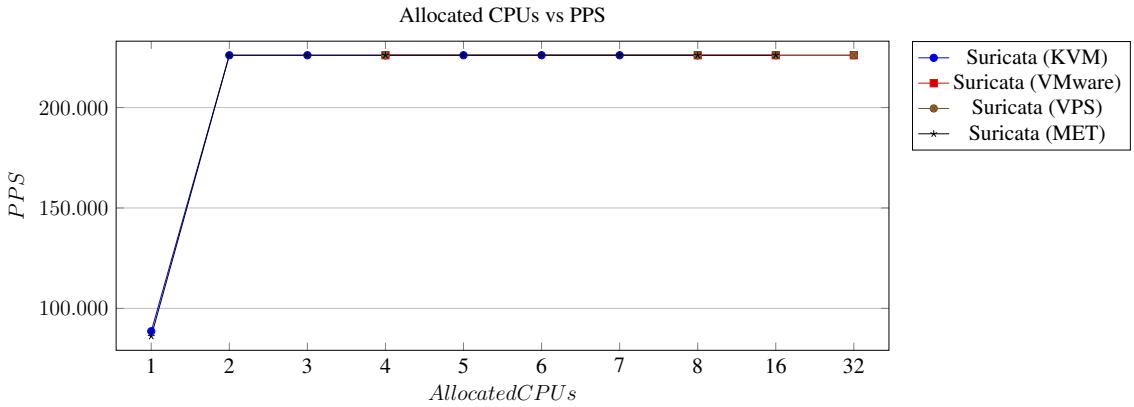

Figure 6.1: Suricata: Allocated CPU vs PPS - (bigFlows.pcap)

With Snort3 we haven't spent too much time, but we wanted to give it a shot, since its promises created great expectations. We tried it in our lab and later introduced it for PCAPs for KVM and the bare-metal context. Its results were a deception as we found it to be constantly performing at the exact same level whenever changing the allocated CPU resources. We've tried to investigate further on its configuration, tried to dig further into limitations at the Southbridge or the Northbridge of the architectures, but the results were very consistent throughout resources' allocation and environment changing (KVM and bare-metal only). No further tuning was attempted as we swiftly ruled it out from our foreseen activities. Unfortunately, being in Alpha, led us away from it, and our findings were reduced to a quick deployment. Yet, such quick approach worked better on the remaining System Under Test (SUT). Still in regards with Snort3, for example in Figure 6.5 we can see how Snort3, even not using the available resources, easily outperforms Snort2 in PCAP processing. Assuming that both are not using more than 1 CPU for processing, in some cases we saw Snort3 performing almost 3 times more PPSs than Snort2, but

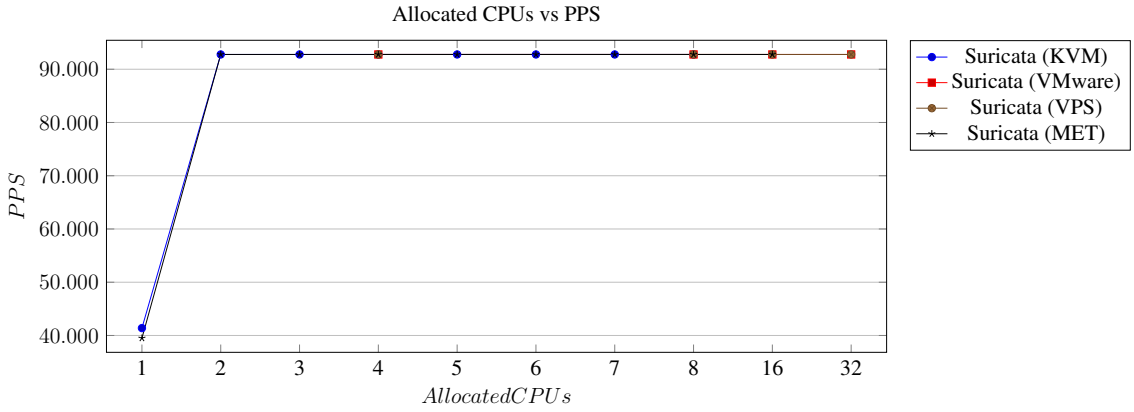

Figure 6.2: Suricata: Allocated CPU vs PPS - (purplehaze.pcap)

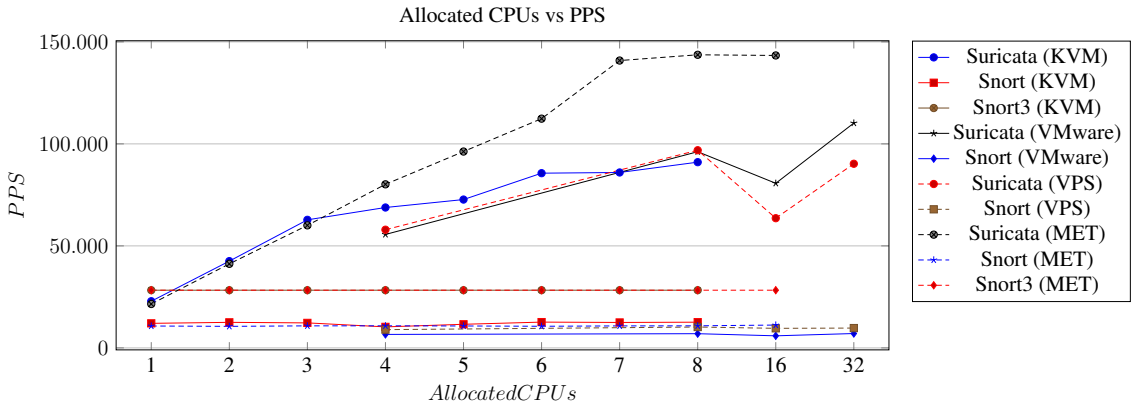

Figure 6.3: Allocated CPU vs PPS - (maccdc2011 00010 ... .pcap)

other times just as much. We also found Snort3 to be consuming more memory whenever more CPUs were allocated. Figure 6.7 shows that either in the industrial bare-metal server or in our home desktop's KVM lab, the IDS increased memory side-by-side and linearly with the number of CPUs. Yet, that did not change the performance for processed packets. It should be noted that the tests consider amounts in the order of the dozens of MB, which are insignificant considering the total amount of available memory. Overall we never found memory to be an issue for any of the tested softwares. Intuition suggested that an improved performance could be achieved by allocating memory differently. Even with huge amounts of memory available, in some of the contexts, this was not seized, and again, the variations are in the order of the dozens of MB. Looking at some memory readings for Suricata, it was interesting to find how memory usage decreases with CPU additions. It seems to imply that, while running on one or few cores, Suricata will use more memory while piping to processors, reducing its usage with more cores available. Nonetheless these values sometimes increased while using a large amount of CPUs (see Figure 6.9).

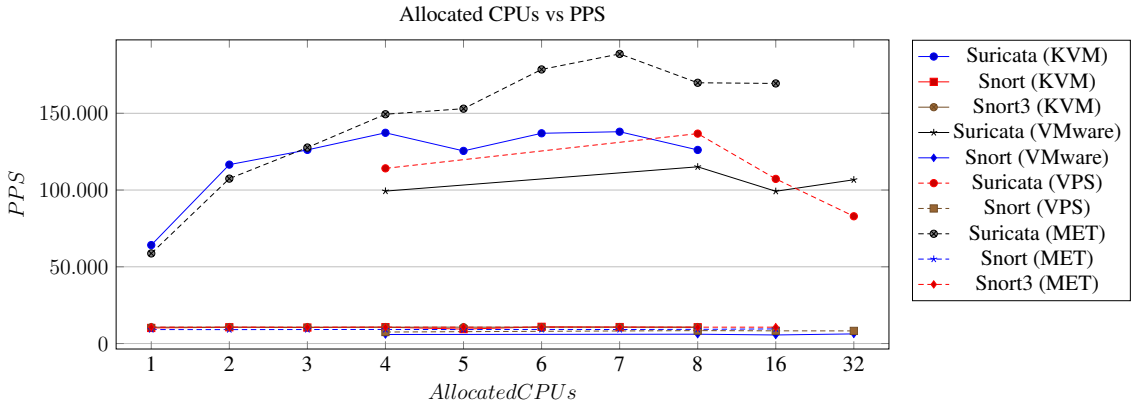

Figure 6.4: Allocated CPU vs PPS - (maccdc2011 00013 ... .pcap)

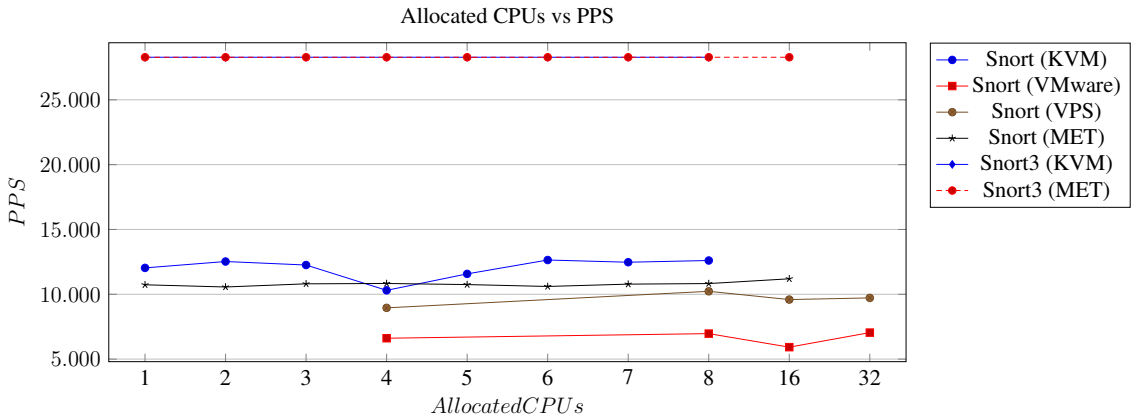

Figure 6.5: Snort2 and Snort3: Allocated CPU vs PPS - (maccdc2011 00010 ... .pcap)

6.2 IDS in pre-production

Pre-production revealed that the amount of traffic handled was considerably higher than the original expectations. As depicted in Figure 4.5 we were expecting to receive around 1Gbps. However results measured more than 3Gbps in core hours, with some peaks around 4.5Gbps. At the same hour, the external interface, measured around 200 and 250Mbps, but in some periods and peaks a maximum around 1Gbps was observed. The initial configuration revealed an incapacity to cope with such an unexpected load by both Snort and Suricata resulting in non-negligible amounts of dropped packets. These drops were not constant, but apparently related to bursts or particular peaks of traffic or sessions. However, without access to other network resources and tools, it was not possible to investigate further or define a pattern for them. To reduce entropy in our tests, we started by aligning the amount of rules in both softwares, as exemplified in Listings 3.1 to 3.4. We also reduced output facilities, and kept syslog only, which was set to pipe to our SIEM in test.

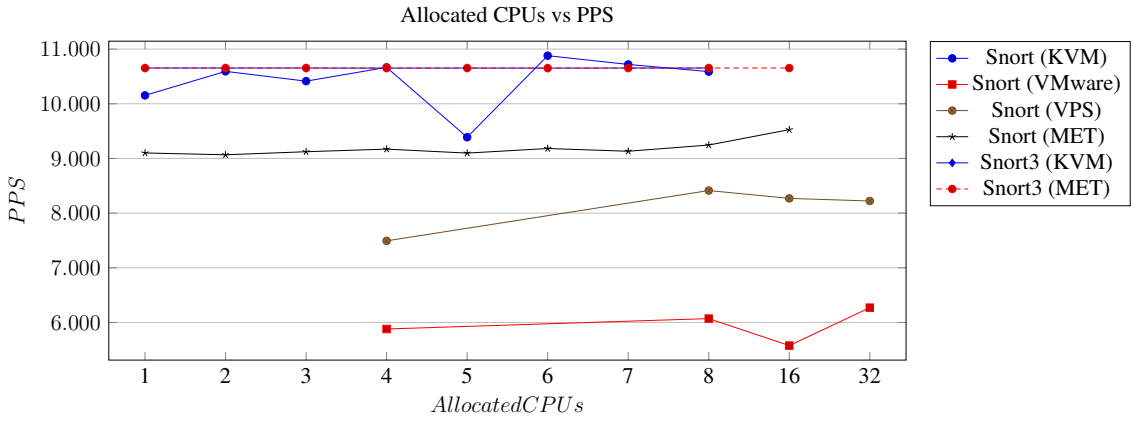

Figure 6.6: Snort2 and Snort3: Allocated CPU vs PPS - (maccdc2011 00013 ... .pcap)

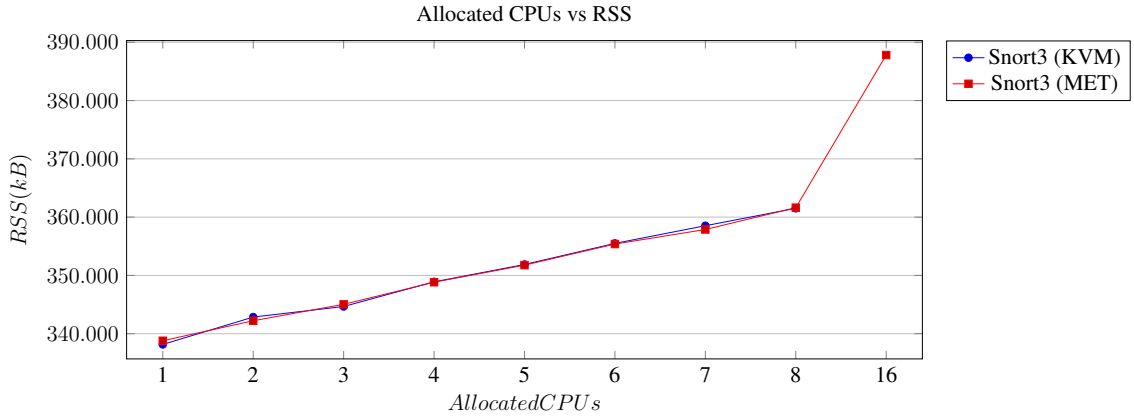

Figure 6.7: Snort3: Allocated CPU vs RSS - (maccdc2011 00010 ... .pcap)

6.2.1 Tuning Suricata

Most of the following configurations can be set either in a Suricata's configuration file or as options and arguments. These are based on Suricata User Guide's [46] Performance chapter and some of its details have already been addressed in section 3.4.

Runmodes

As discussed in section 3.4, Suricata allows different options for each packet acquisition method. Although the default is *autofp* (auto flow pinned load balancing), the manual points to mode *workers* to be preferable for scenarios with stringent performance requirements. Gains are justified by *workers* ability to distribute the packets over various threads. However, this wasn't a success in our experiment as we still had a very significant amount of packet drops.

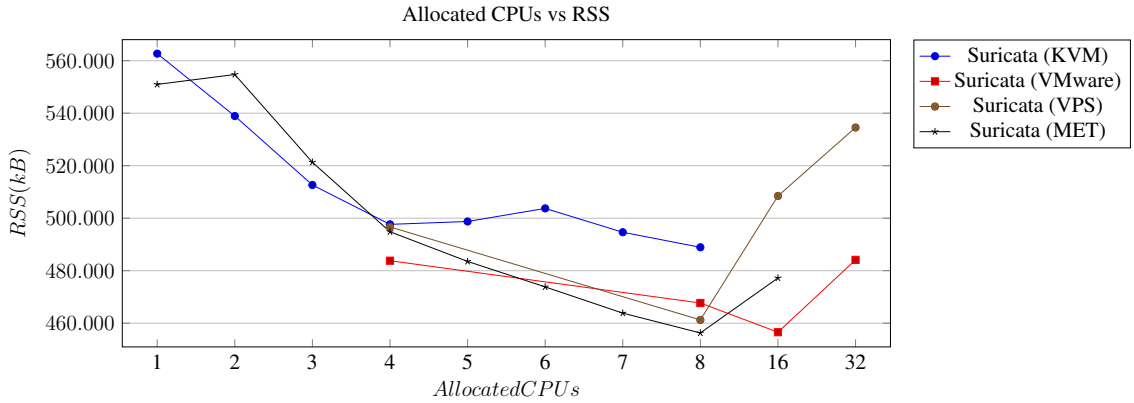

Figure 6.8: Suricata: Allocated CPU vs RSS - (maccdc2011 00013 ... .pcap)

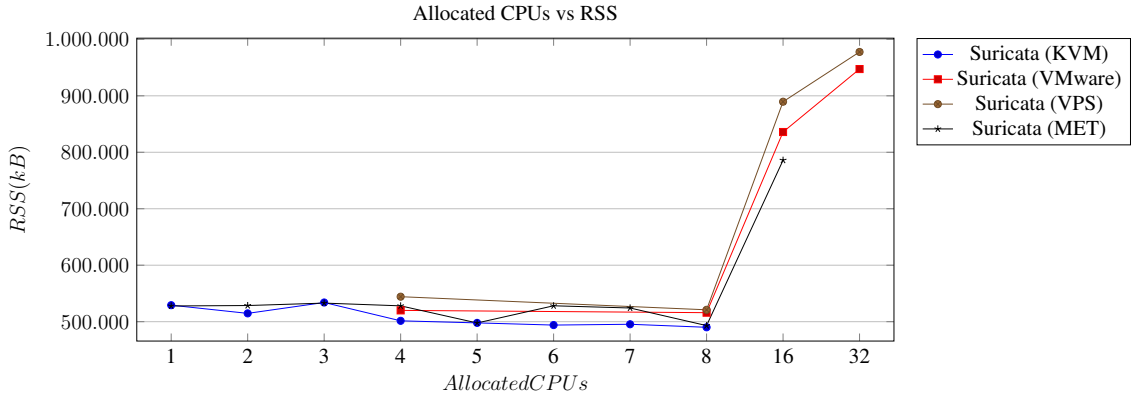

Figure 6.9: Suricata: Allocated CPU vs RSS - (maccdc2011 00010 ... .pcap)

AF_PACKET Capture method

Tuning of the AF_PACKET (Linux high speed) capture method was attempted by increasing the number of threads. Nonetheless the *auto* value uses the number of cores and therefore changing this value hasn't improved the results. For both AF_PACKET and PF_RING, about which we will speak in the following section, there are some configuration options for *cluster-type*. The recommended mode is *cluster_flow* which is the default. Yet we perceived a possible decrease in dropped packets using *cluster_cpu*. Since these experiments were being conducted in live traffic captures, with one running instance only, we couldn't extract detailed information to prove such perceived results. We also tried techniques around options, such as *use-mmap*, *ring-size*, *max-pending-packets* or *sgh-mpm-context*, but with no success in reducing the drops to zero.

PF_RING Capture method

To proceed with PF_RING we had to install and configure it from ntop's repositories, after checking its dependencies. Suricata was recompiled and reinstalled with PF_RING enabled. We reviewed the options left from the AF_PACKET approach. It worths men-

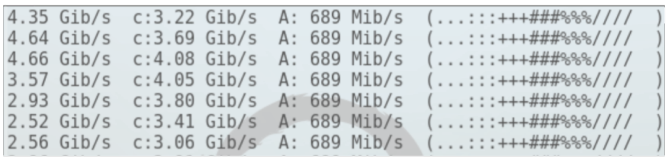

Figure 6.10: Internal network traffic measured with speedometer 2.8

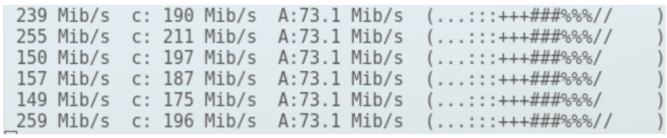

Figure 6.11: External network traffic measured with speedometer 2.8

tioning the following details, also mentioned as tuning considerations in the relevant section of Suricata's Guide [46]. *max-pending-packets* was left at its default with the value of 1024, we found its increase to be of no difference in our setup. *mpm-algo* (pattern matcher algorithm's control) was left in *auto* which should default to *AC* (Aho–Corasick string-searching algorithm) since we hadn't include Hyperscan (a high-performance multiple regex matching library). *detect.profile* was left as medium and no *custom-values* were changed, however these could have been used to increase performance at the cost of memory. *detect.sgh-mpm-context* (multi pattern matcher) was set to *full*, which with *AC* in *mpm-algo* could lead to a huge memory usage, although this was not observed (we've seen less than 9GB of RAM being used most of the time). In PF_RING's configuration we've set one thread as more than one is still experimental and for *cluster-type* we opted for *cluster_round_robin* over the suggested *cluster_flow* since we found it to perform better in our setup.

Listing 6.1: (Excerpts) Starting Suricata at eth0

```
1  root@host:~# suricata -c /etc/suricata/suricataf0.yaml -vvv --pfring-int=eth0 --
     ↪ pfring-cluster-id=98 --pfring-cluster-type=cluster_round_robin
```

With this setup we finally got the desired results of zero packets dropped. In our longest run, we were able to have two concurrent instances running side-by-side, one for each interface, for more than 8 days (695836.625s), with zero dropped packets. Those outputs can be seen in Listings 6.2 and 6.3.

Listing 6.2: (Excerpts) Stopping Suricata at eth0

```
182  14/MM/YYYY -- 04:50:00 - <Info> - time elapsed 695836.625s
191  14/MM/YYYY -- 04:50:00 - <Notice> - Stats columnsfor 'eth0':  pkts: 22303999950,
   ↪ drop: 0 (0.00%), invalid chksum: 0
```

Listing 6.3: (Excerpts) Stopping Suricata at eth1

```
182  14/MM/YYYY -- 04:50:02 - <Info> - time elapsed 695824.375s
191  14/MM/YYYY -- 04:50:02 - <Notice> - Stats columnsfor 'eth1':  pkts: 85710399157,
   ↪ drop: 0 (0.00%), invalid chksum: 0
```

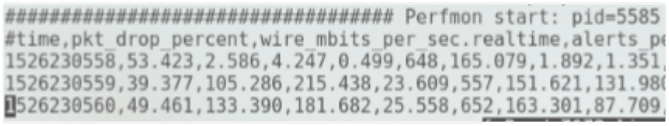

Figure 6.12: Snort's stats showing drops

More details on starting and stopping those tasks can be seen in Listings 6.4 and 6.5 or in the appendices in G.

Listing 6.4: (Excerpts) Starting and stopping Suricata at eth0

```
  1  root@host:~# suricata -c /etc/suricata/suricataf0.yaml -vvv --pfring-int=eth0 --
       ↪ pfring-cluster-id=98 --pfring-cluster-type=cluster_round_robin
  2  6/MM/YYYY -- 03:32:37 - <Notice> - This is Suricata version 4.0.4 RELEASE
  3  6/MM/YYYY -- 03:32:37 - <Info> - CPUs/cores online: 16
 84  6/MM/YYYY -- 03:32:40 - <Info> - 38 rule files processed. 12529 rules successfully
     ↪ loaded, 0 rules failed
135  6/MM/YYYY -- 03:32:40 - <Info> - 12534 signatures processed. 1158 are IP-only rules
     ↪ , 5309 are inspecting packet payload, 7656 inspect application layer, 0 are
     ↪ decoder event only
167  6/MM/YYYY -- 03:32:43 - <Info> - fast output device (regular) initialized: fast0.
     ↪ log
168  6/MM/YYYY -- 03:32:43 - <Info> - stats output device (regular) initialized: statsf0
     ↪ .log
169  6/MM/YYYY -- 03:32:43 - <Config> - AutoFP mode using "Hash" flow load balancer
170  6/MM/YYYY -- 03:32:43 - <Info> - Using round-robin cluster mode columnsfor PF_RING
     ↪ (iface eth0)
171  6/MM/YYYY -- 03:32:43 - <Info> - Going to use 1 ReceivePfring receive thread(s)
172  6/MM/YYYY -- 03:32:43 - <Perf> - (RX#01) Using PF_RING v.7.0.0, interface eth0,
     ↪ cluster-id 98, single-pfring-thread
173  6/MM/YYYY -- 03:32:43 - <Info> - RunModeIdsPfringAutoFp initialised
174  6/MM/YYYY -- 03:32:43 - <Config> - using 1 flow manager threads
175  6/MM/YYYY -- 03:32:43 - <Config> - using 1 flow recycler threads
176  6/MM/YYYY -- 03:32:43 - <Info> - Running columnsin live mode, activating unix
     ↪ socket
177  6/MM/YYYY -- 03:32:43 - <Info> - Using unix socket file '/var/run/suricata/suricata
     ↪ -columnscommand.socket'
178  6/MM/YYYY -- 03:32:43 - <Notice> - all 17 packet processing threads, 4 management
     ↪ threads initialized, engine started.
179  ^C
180  14/MM/YYYY -- 04:50:00 - <Notice> - Signal Received.  Stopping engine.
181  14/MM/YYYY -- 04:50:00 - <Perf> - 0 new flows, 0 established flows were timed out,
     ↪ 0 flows columnsin closed state
182  14/MM/YYYY -- 04:50:00 - <Info> - time elapsed 695836.625s
183  14/MM/YYYY -- 04:50:00 - <Perf> - 232394857 flows processed
184  14/MM/YYYY -- 04:50:00 - <Perf> - (RX#01) Kernel: Packets 22303999950, dropped 0
185  14/MM/YYYY -- 04:50:00 - <Perf> - (RX#01) Packets 22303999950, bytes 14159420517052
186  14/MM/YYYY -- 04:50:00 - <Perf> - AutoFP - Total flow handler queues - 16
187  14/MM/YYYY -- 04:50:00 - <Info> - Alerts: 28454771
188  14/MM/YYYY -- 04:50:00 - <Perf> - ippair memory usage: 398144 bytes, maximum:
     ↪ 16777216
189  14/MM/YYYY -- 04:50:00 - <Perf> - host memory usage: 549784 bytes, maximum:
     ↪ 33554432
190  14/MM/YYYY -- 04:50:00 - <Info> - cleaning up signature grouping structure...
     ↪ columnscomplete
191  14/MM/YYYY -- 04:50:00 - <Notice> - Stats columnsfor 'eth0':  pkts: 22303999950,
     ↪ drop: 0 (0.00%), invalid chksum: 0
```

Listing 6.5: (Excerpts) Starting and stopping Suricata at eth1

```
1  root@host:~# suricata -c /etc/suricata/suricataf1.yaml -vvv --pfring-int=eth1 --
     ↪ pfring-cluster-id=99 --pfring-cluster-type=cluster_round_robin
2  6/MM/YYYY -- 03:32:51 - <Notice> - This is Suricata version 4.0.4 RELEASE
3  6/MM/YYYY -- 03:32:51 - <Info> - CPUs/cores online: 16
```

```
84  6/MM/YYYY -- 03:32:54 - <Info> - 38 rule files processed. 12529 rules successfully
    ↪ loaded, 0 rules failed
135  6/MM/YYYY -- 03:32:54 - <Info> - 12534 signatures processed. 1158 are IP-only rules
    ↪ , 5309 are inspecting packet payload, 7656 inspect application layer, 0 are
    ↪ decoder event only
167  6/MM/YYYY -- 03:32:57 - <Info> - fast output device (regular) initialized: fast1.
    ↪ log
168  6/MM/YYYY -- 03:32:57 - <Info> - stats output device (regular) initialized: statsf1
    ↪ .log
169  6/MM/YYYY -- 03:32:57 - <Config> - AutoFP mode using "Hash" flow load balancer
170  6/MM/YYYY -- 03:32:57 - <Info> - Using round-robin cluster mode columnsfor PF_RING
    ↪ (iface eth1)
171  6/MM/YYYY -- 03:32:57 - <Info> - Going to use 1 ReceivePfring receive thread(s)
172  6/MM/YYYY -- 03:32:57 - <Perf> - (RX#01) Using PF_RING v.7.0.0, interface eth1,
    ↪ cluster-id 99, single-pfring-thread
173  6/MM/YYYY -- 03:32:57 - <Info> - RunModeIdsPfringAutoFp initialised
174  6/MM/YYYY -- 03:32:57 - <Config> - using 1 flow manager threads
175  6/MM/YYYY -- 03:32:57 - <Config> - using 1 flow recycler threads
176  6/MM/YYYY -- 03:32:57 - <Info> - Running columnsin live mode, activating unix
    ↪ socket
177  6/MM/YYYY -- 03:32:57 - <Info> - Using unix socket file '/var/run/suricata/suricata
    ↪ -columnscommand.socket'
178  6/MM/YYYY -- 03:32:57 - <Notice> - all 17 packet processing threads, 4 management
    ↪ threads initialized, engine started.
179  ^C
180  14/MM/YYYY -- 04:50:02 - <Notice> - Signal Received.  Stopping engine.
181  14/MM/YYYY -- 04:50:02 - <Perf> - 0 new flows, 0 established flows were timed out,
    ↪ 0 flows columnsin closed state
182  14/MM/YYYY -- 04:50:02 - <Info> - time elapsed 695824.375s
183  14/MM/YYYY -- 04:50:02 - <Perf> - 143678837 flows processed
184  14/MM/YYYY -- 04:50:02 - <Perf> - (RX#01) Kernel: Packets 85710399157, dropped 0
185  14/MM/YYYY -- 04:50:02 - <Perf> - (RX#01) Packets 85710399157, bytes 78563883813003
186  14/MM/YYYY -- 04:50:02 - <Perf> - AutoFP - Total flow handler queues - 16
187  14/MM/YYYY -- 04:50:02 - <Info> - Alerts: 2388359
188  14/MM/YYYY -- 04:50:02 - <Perf> - ippair memory usage: 398144 bytes, maximum:
    ↪ 16777216
189  14/MM/YYYY -- 04:50:02 - <Perf> - host memory usage: 398144 bytes, maximum:
    ↪ 33554432
190  14/MM/YYYY -- 04:50:02 - <Info> - cleaning up signature grouping structure...
    ↪ columnscomplete
191  14/MM/YYYY -- 04:50:02 - <Notice> - Stats columnsfor 'eth1':  pkts: 85710399157,
    ↪ drop: 0 (0.00%), invalid chksum: 0
```

Though we found a setup that can deal with the actual measured demand, we still had options to tune further on the high performance configuration. Hyperscan was an option not tested with Suricata that deserves some attention in the future. If more traffic is to be added, *detect.profile* can be changed accordingly. As seen in Suricata's documentation, a possible custom profile for high performance can include the base settings shown in 6.6

Listing 6.6: High performance *detect.profile*

```
detect:
  profile: custom
  custom-values:
      toclient-groups: 200
      toserver-groups: 200
  sgh-mpm-context: auto
  inspection-recursion-limit: 3000
```

However, this can increase the memory consumptions, depending on the amount of rules loaded. Yet the trade off can be managed adding CPUs instead, which can be cheaper for

such hardware. In our specific environment, we believe that CPUs can be stressed further and we still saw memory available, so there should be enough resources to increase the load. It is also possible that such customization extends the loading times of rules' sets. Finally, to achieve one of the project's requirements we've set separate configuration and log files for each instance, so those could be managed separately. And we've also managed to send those outputs separately, through syslog, to the external log collector and SIEM.

Note: For more details on Statistics and Detecting Packet Loss we recommend the reading of the relevant section at http://suricata.readthedocs.io/en/latest/performance/statistics.html

6.2.2 Tuning Snort

While tuning Snort, we found less parameters to be pre-set in the configuration file, but similar options and arguments to start the application with, apart from the ones related with multi-threading.

AF_PACKET Capture method

Since the beginning it seemed clear that, trying to take the most out of AF_PACKET wouldn't be enough. Packet drops were already between 50 and 90%, and a singled core technique surelly wouldn't suffice. Because this capture method was also not successful with Suricata, we soon decided to move to the PF_RING method.

PF_RING Capture method

Again, with Snort, we had to install and configure PF_RING from ntop's repositories. As in Snort's installation also this setup was not straight forward, but, eventually worked after a thorough review of the configuration steps. Once Data Acquisition library (DAQ) was correctly setup and PF_RING module loaded, everything started to work as expected. Snort was set to start with *–daq-dir*, *–daq pfring* and *–daq-mode passive* (IDS mode). We noticed a slight increase in the performance but yet drops were rather constant. Looking further into its options, *–daq pfring* allows to set a *clusterid*. With this value set, PF_RING distributes the load across multiple processes. Using this technique against our external interface didn't result in zero drops, not even when adding as much as 16 instances. In some of these tests, packet drops were above 80% despite of some regular readings at 0%. Moving forward we tried clustering plus core binding with *–daq-var bindcpu=n*. At this stage we started to look at our internal interface, the most loaded one. With this technique, and adding as much as 16 CPUs, we achieved losses around 23%; again with regular readings at 0%. Because the summary is independent per instance, we had to run tests without

daemon options so we may look at each report summary while closing the application, otherwise we were liaising on the first instance's summary only. During those tests, we found warnings such as *session exceeded configured max bytes* or *SMTP memcap exceeded* which we tackled adding as much memory as possible to relevant preprocessors: *stream5_global: memcap 1073741824*, *stream5_tcp: max_queued_bytes 1073741824* or *smtp: memcap 104857600*. Yet, with 8 instances, for a short session, 5 reported zero drops, 1 reported 22% and another 0.2%, and these were not the ones receiving more packets. Trying a 19 hours session against our internal listening interface, with 16 instances, again resulted in just some reporting zero drops. We saw this session using as much as 10GB of RAM and reporting awkward results. As seen in 6.7, some instances reported to have received multiple times as much as others, and some reported to have dropped far more packets that the ones received and analyzed.

Listing 6.7: Snort's dropped examples in 3 selected concurrent instances

```
Packet I/O Totals:
   Received:    103031437
   Analyzed:    103031437 (100.000%)
    Dropped:    329464135 ( 76.177%)
Packet I/O Totals:
   Received:    270830939
   Analyzed:    270830939 (100.000%)
    Dropped:    154232918 ( 36.285%)
Packet I/O Totals:
   Received:    435857038
   Analyzed:    435857038 (100.000%)
    Dropped:    658235286 ( 60.163%)
```

Moving forward with Snort's performance tuning, core insulation with Clustering plus Core Binding technique, seemed to be the most "performant" approach. This technique, against the internal interface, seemed to work with minimal losses, at least out of business hours. With the purpose of establishing a comparison with Suricata (which was able to perform flawlessly against both listening interfaces for several days), we started by assigning 8 (clustered+binded) CPUs for each interface in a total of 16 allocated. This resulted in both constantly reporting drops. Already at this stage we struggled with foreseeing how to manage such baffling logging. Each Clustered+Core Binded instance requires a dedicated log folder. As such, moving to the 16 plus 16 instances attempt (trying to equate Suricata's scenario), resulted in a huge amount of logs to be processed and managed. In a short 3 minutes session, 5 out of 16 instances reported drops at the external listening interface, with values from 2 to 70%; remaining 16, reported quite similarly. With such results, out of core/business hours, we haven't tried to deal with such amount of logs to ship outside the IDS. It seemed clear that, the amount of traffic that our experiment is already trying to address, with such hardware, is already overwhelming for the tested techniques.

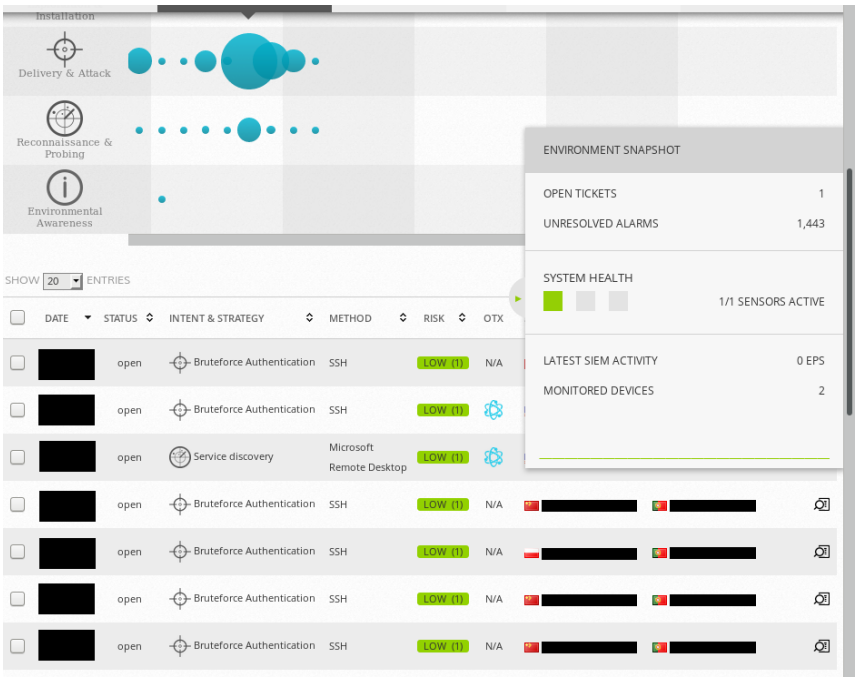

Figure 6.13: OSSIM showing a list of alarms

Yet we could have looked deeper into options such as PF_RING ZC or PF_RING DNA available since 2012, if compatible with our setup. The latter DAQ has a fee, but universities and research institutions might be able to get it at no cost as stated by ntop. This should be considered for future work and certainly if the host organization decides for Snort rather than Suricata. Already in core hours, in a new 19 hours session, all instances reported drops, from 11 to 82%, while processing against our internal listening interface.

6.3 SIEM - OSSIM

Since its initial implementation in our laboratory, we found OSSIM to hung if enough resources were not available, yet after setting minimum decent values, it performed flawlessly. OSSIM frequently displays messages highlighting that new updates are available. From one side it can be a pain for the system's administrator to keep the most up to date version, but from the other side it indicates that the product is maintained and kept up to date, either with software patches or with new signatures. OSSIMs dashboards and events' searches and filters seem to accomplished what the host organization should need for this piece of the implementation. Nonetheless it would need a further assessment since the existing, customized, solutions might already deliver part of what this one offers, and therefore there would be a new adaptation to the teams involved. Because of its integration capabilities, we believe that this tool or a similar one is definitely a must, and suits perfectly, to perform as an interface to IDSs alerts. Unfortunately we didn't create a framework to assess this tool as this wasn't part of our project's objectives.

Chapter 7

Conclusion

In this work we were able to prove the possibility of building a security appliance with commodity hardware, relying entirely on Free/Libre and Open Source Software. We were able to try state-of-the-art Intrusion Detection Systems in a simple home or small office network, but also on a complex and highly dense infrastructure. We've tried single and multiple instances, auto start customization, rules' update managers, multiple listening interfaces, output shipping, log and alert collection and also event correlation. This work allowed, not only, to develop knowledge in the field of such tools, but also responsiveness while analyzing such scenarios. While we were developing this work, we had contact with similar implementations, in parallel professional projects, based on commercial dedicated appliances (built in firewalls), which enabled a balance between approaches.

Our focus was on comparing Snort and Suricata as tools to implement a security solution. Soon we found their similarities, but also their differences, in their architecture, configuration, management, customization and interaction. At an earlier stage, with low resources and low demand, we were minded to see Snort as the way to go with. Yet, after looking into huge differences in performance, we understood that we couldn't scale Snort to reach Suricata's performance within our setup and with the demand observed at the host organization. Specially in the final stages of our work, we found Suricata to largely outperform Snort. Every tuning technique we found documented simply wasn't enough to reduce Snort's drops to zero.

The plan was to assess the IDSs so we may understand which one to take to production, acting as IPS allied with a Linux based firewall. Moving such approach to IPS, with even a reduced number of dropped packets, in such a complex and demanding infrastructure, is not acceptable. Therefore we concluded that only Suricata has proven to be capable of such task within our setup and using such hardware. With Suricata performing as wished, our focus left the Snort approach behind, in spite of the several tuning techniques attempted to increase performance.

We still believe that both products are reliable enough and deserve to be assessed within some boundaries and expectations. Both are highly customizable and can adapt to the available hardware, presenting multiple techniques from end to end.

Regarding Snort, we believe that we left it short while testing the offline PCAP approach, we later found some techniques that can be used in such operation mode. Nonetheless, its performance differences would be difficult to stretch towards its opponent. We found it to be more complex to deploy with a huge amount of dependencies and customization steps till its first run. This complexity seems to have been slightly addressed in its newer Alpha version, that seemed more simple to deploy. It is important to say that Snort is largely documented and we had no problems finding answers within the available documentation. This feeling should also be supported by the fact that the product hasn't change much along the years. In such FLOSS approaches, it is important to be backed by the community, and we had the chance to interact with Snort communities through several lines of contacts, which we found to be very helpful, prompt and willing to collaborate. There is a live IRC channel in freenode with constant people willing to share knowledge and experience. We had the opportunity to discuss some details with the IRC community, to report a false positive on Snort's website, report an error on the available documentation and interact as a registered user. This gave us a very inviting experience and a high level of confidence which was already sponsored by their experience in this field.

With Suricata, as Rødfoss [38], we also found it to be easier to setup and configure. This led us to think that maybe Snort has stopped in time, in the sense that some possible improvements are not developed anymore. Suricata, besides being easier to set up, also seemed very customizable, as its creation aims it to be able to effortlessly integrate with several sorts of external softwares and deployable in different sorts of hardware. Suricata is highly modular and even its documentation is found in such fashion, with several short guides addressing each topic in short steps till its goal. Being owned and supported by a non-profit foundation, its community-driven development attitude is keeping this project alive and growing followers by the day.

Looking at both, we found them not to be very RAM consuming, which would have been a resource to worry about. As Eugene in 2011 [2], we also found both to normally use less than 1GB of RAM in low demand and never above $\sim$10GB while running in a highly demanding environment. As sometimes advised, if necessary, we'd rather scale CPU over RAM which in some scenarios can also be more inexpensive. Both require a learning and adaptation period to administer. Specially, while deploying, it is very important to customize details regarding the assets within the network to secure, their roles and

the way they will interact. A non-customized setup can lead to an important increase of wasted resources. Rules' management also requires such dedication, during the setup, but also maintaining it. Threats keep changing, assets keep updating, protocols are changed and so do rules; so even if the users don't change their habits, everything else requires maintenance. As stated by Khalil [24], we also agree that a team's effort is necessary to keep such system correctly administered and maintained. Automatic/unattended rules updates are not recommended and a manual approach should be more suitable in critical and demanding environments. Otherwise, resources can be wasted (e.g. unnecessary rules being checked) and false positives can increase (e.g. leading to spam and overwhelming alarms).

In our implementation we found PF_RING to play a key role in the performance. This high speed packet capture socket, largely improved our capacity and proved other approaches to be incompatible with our setup. Moreover, PF_RING still keeps being developed and presented in the relevant forums, boosting its capacity in each development iteration. We slightly tried its Zero Copy (ZC) framework with Snort (without success), and would like to have tried their Direct NIC Access (DNA), to understand if Snort could have been scaled higher in that sense, after we failed scaling everything else at our hands. PF_RING should also be preferable if the IDS is deployed upon an hypervisor, but we haven't tested it because our setups didn't required it.

That said, with the information available and the presented assumptions and requirements, we believe that the host organization should consider in fact to take such approach to production. We have proven that it is possible to deploy the same level of service using such hardware and that, with some customization, it is possible to increase the type of results taken from the inspection performed. Nonetheless, either as IDS or IPS (the latter should be staged and customized as IDS in advance), we believe that this deployment requires a plan for implementation which includes an adaptation window for both the system and its administrators, providing such critical infrastructure with a high level of confidence. Apart from this learning phase, we believe that other systems' administrators can swiftly keep up with managing such tool, mastering the topic in just a couple of months.

This page intentionally left blank.

Appendix A

Snort

A.1 Snort basic setup

Note: This section is based on *Snort 2.9.9.x on Ubuntu 14 and 16* by Noah Dietrich from https://www.snort.org/documents

Install *tcpdump*.

```
sudo apt-get install tcpdump
```

Install *less*.

```
sudo apt-get install less
```

Activate interface on bridge.

```
sudo ip link set eth0 up
```

Set interface to be persistent.

```
sudo nano /etc/network/interfaces
```

```
auto eth0
iface eth0 inet manual
        up ifconfig eth0 up
```

Check activity at the interface.

```
sudo tcpdump -i eth0
```

Install build-essential.

```
sudo apt-get install build-essential
```

Create a snort directory.

```
mkdir ~/snort_src
cd ~/snort_src
```

Install CA certificates for the necessary downloads.
Not advisable alternative using *wget –no-check-certificate*.

```
sudo apt-get install ca-certificates
```

Download latest version of daq.

```
wget https://www.snort.org/downloads/snort/daq-2.0.6.
   ↪ tar.gz
```

Check md5 as in https://www.snort.org/downloads/snort/md5s.
67b915f10c0cb2f7554686cdc9476946 daq-2.0.6.tar.gz

```
md5sum -c <<<"67b915f10c0cb2f7554686cdc9476946 daq
   ↪ -2.0.6.tar.gz"
```

Download latest version of snort.

```
wget https://www.snort.org/downloads/snort/snort
   ↪ -2.9.11.1.tar.gz
```

Check md5 as in https://www.snort.org/downloads/snort/md5s.
378e3938b2b5c8e358f942d0ffce18cc snort-2.9.11.1.tar.gz

```
md5sum -c <<<"378e3938b2b5c8e358f942d0ffce18cc snort
   ↪ -2.9.11.1.tar.gz"
```

Install dependencies.
Note: Without libpcap-dev configure will throw *ERROR! Libpcap library version >= 1.0.0 not found. Get it from* http://www.tcpdump.org.

```
sudo apt-get install bison flex
sudo apt-get install libpcap-dev
```

Extract, configure, make and install daq.

```
tar -xvzf daq-2.0.6.tar.gz
cd daq-2.0.6
./configure
make
sudo make install
```

Install dependencies.
Note: Without libpcre3-dev configure will throw *ERROR! Libpcre header not found. Get it from* http://www.pcre.org.
Note: Without libdumbnet-dev configure will throw *ERROR! dnet header not found, go get it from* http://code.google.com/p/libdnet/ *or use the –with-dnet-* options, if you have it installed in an unusual place*
Note: Without zlib1g-dev configure will throw *ERROR! zlib header not found, go get it from* http://www.zlib.net

```
sudo apt-get install libpcre3-dev
sudo apt-get install libdumbnet-dev
sudo apt-get install zlib1g-dev
```

Extract, configure, make and install snort.

```
cd ..
tar xvfz snort-2.9.11.1.tar.gz
cd snort-2.9.11.1
./configure --enable-sourcefire
make
sudo make install
```

Create the snort group and user.

```
sudo groupadd snort
sudo useradd snort -r -s /sbin/nologin -c SNORT_IDS -g
   ↪  snort
```

Create the Snort directories.
Configurations and rules: */etc/snort*
Compiled rules (.so rules): */usr/local/lib/snort_dynamicrules*

```
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
```

Create files to store rules and ip lists.

```
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
```

Create the logging directories.
Alerts: */var/log/snort*

```
sudo mkdir /var/log/snort
sudo mkdir /var/log/snort/archived_logs
```

Set permissions.

```
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
```

Set ownership on folders.

```
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
```

Copy configuration files and dynamic preprocessors.
classification.config
file_magic.conf
reference.config
snort.conf
threshold.conf
attribute_table.dtd
gen-msg.map
unicode.map

```
cd ~/snort_src/snort-2.9.11.1/etc/
sudo cp *.conf* /etc/snort
sudo cp *.map /etc/snort
sudo cp *.dtd /etc/snort
cd ~/snort_src/snort-2.9.11.1/src/dynamic-
   ↪ preprocessors/build/usr/local/lib/
   ↪ snort_dynamicpreprocessor/
sudo cp * /usr/local/lib/snort_dynamicpreprocessor/
```

Directories summary:

Snort binary file:	*/usr/local/bin/snort*
Snort configuration file:	*/etc/snort/snort.conf*
Snort log data directory:	*/var/log/snort*
Snort rules directories:	*/etc/snort/rules*
	/etc/snort/so_rules
	/etc/snort/preproc_rules
	/usr/local/lib/snort_dynamicrules
Snort IP list directories:	*/etc/snort/rules/iplists*
Snort dynamic preprocessors:	*/usr/local/lib/snort_dynamicpreprocessor/*

Snort directory tree:

```
tree /etc/snort/
```

Output:

```
/etc/snort/
|-- attribute_table.dtd
|-- classification.config
|-- file_magic.conf
|-- gen-msg.map
|-- preproc_rules
|-- reference.config
|-- rules
|       |-- iplists
|       |       |-- black_list.rules
|       |       |-- white_list.rules
|       |-- local.rules
|-- sid-msg.map
|-- snort.conf
|-- so_rules
|-- threshold.conf
|-- unicode.map
```

Backup *snort.conf* file.

```
sudo cp /etc/snort/snort.conf /etc/snort/snort.conf.
   ↪ bak
```

Comment out all referenced configurations files.
Note: We will use PulledPork to manage rulesets instead.

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

Edit Snort configuration file.

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

Edit Line 45 setting the internal network.

```
ipvar HOME_NET xx.xx.xx.xx/xx
```

Set file paths at Line 104 and 113.

```
var RULE_PATH /etc/snort/rules
var SO_RULE_PATH /etc/snort/so_rules
var PREPROC_RULE_PATH /etc/snort/preproc_rules
var WHITE_LIST_PATH /etc/snort/rules/iplists
var BLACK_LIST_PATH /etc/snort/rules/iplists
```

Uncomment Line 546 to enable *local.rules*.
Note: Helpful for troubleshooting and/or customization.

```
include $RULE_PATH/local.rules
```

Test Snort configuration files.

-T test the configuration file
-c configuration file path
-i interface to listen on

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

Create test rule.

```
sudo nano /etc/snort/rules/local.rules
```

```
alert icmp any any -> $HOME_NET any (msg:"ICMP test
   ↪ detected"; GID:1; sid:10000001; rev:001;
   ↪ classtype:icmp-event;)
```

Add meta-information for compatibility with Barnyard2 and PulledPork.

```
sudo nano /etc/snort/sid-msg.map
```

```
#v2
1 || 10000001 || 001 || icmp-event || 0 || ICMP Test
   ↪ detected || url,tools.ietf.org/html/rfc792
```

Test Snort configuration files.

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

Run Snort in NIDS mode with output to the console.

-A console	prints fast mode alerts to stdout
-q	quiet mode, no banner
-u snort	user
-g snort	group
-c /etc/snort/snort.conf	configuration file path
-i ens4	interface to listen on

```
sudo /usr/local/bin/snort -A console -q -u snort -g
   ↪ snort -c /etc/snort/snort.conf -i eth0
```

Note: Logs are saved to */var/log/snort* as snort.log.xxxxxxxxxx

A.1.1 Barnyard2

Barnyard2 allows to get Snort's binary events output to a MySQL/MariaDB database, rather than console or text files.

Install Barnyard2 pre-requisites.

```
sudo apt-get install mariadb-server libmariadbclient-
   ↪ dev mariadb-client autoconf libtool libmariadb-
   ↪ dev-compat
```

Edit Snort configuration file to configure output plugin for *unified2* (binary).

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

Add at Line 522.

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

Note: Logs are saved as snort.u2.xxxxxxxxxx (Unix Epoch time) with 128MB each.

Download latest version of Barnyard2.

```
cd ~/snort_src
```

```
wget https://github.com/firnsy/barnyard2/archive/
   ↪ master.tar.gz -O barnyard2-Master.tar.gz
```

Extract Barnyard2.

```
tar zxvf barnyard2-Master.tar.gz
cd barnyard2-master
autoreconf -fvi -I ./m4
```

Create a soft link from *dnet.h* to *dubmnet.h* as expected by Barnyard2.

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

Configure, make and install.

```
./configure --with-mysql --with-mysql-libraries=/usr/
   ↪ lib/x86_64-linux-gnu
make
sudo make install
```

Test.

```
/usr/local/bin/barnyard2 -V
```

Create Barnyard2 files and directories.

```
sudo cp ~/snort_src/barnyard2-master/etc/barnyard2.
   ↪ conf /etc/snort/
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
```

Note: */var/log/barnyard2* doesn't seem to be used but it will throw an error without it.

Create database and user.

```
sudo mysql -u root -p
create database snort;
use snort;
source ~/snort_src/barnyard2-master/schemas/
   ↪ create_mysql
CREATE USER 'snort'@'localhost' IDENTIFIED BY '
   ↪ dbsnortpass';
grant create, insert, select, delete, update on snort
   ↪ .* to 'snort'@'localhost';
exit
```

Edit Barnyard2 configuration file to setup connection to the database.

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

Add at the end of the file.

```
output database: log, mysql, user=snort password=
   ↪ dbsnortpass dbname=snort host=localhost sensor
   ↪ name=sensor01
```

Change file to hide password in clear.

```
sudo chmod o-r /etc/snort/barnyard2.conf
```

Test Snort in alert mode for binary output.

```
sudo /usr/local/bin/snort -q -u snort -g snort -c /etc
   ↪ /snort/snort.conf -i eth0
```

Check for new snort.u2.xxxxxxxxxx file.

```
ls -lh /var/log/snort/
```

Test Barnyard2.

-c	/etc/snort/barnyard2.conf	configuration file path
-d	/var/log/snort	binary input path
-f	snort.u2	input filename
-w	/var/log/snort/barnyard2.waldo	wlado bookmark file path
-u	snort	user
-g	snort	group

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

Check database for Barnyard2 new entries.

```
mysql -u snort -p -D snort -e "select count(*) from
   ↪ event"
```

A.1.2 PulledPork

PulledPork is a perl script that will download and update snort rulesets, instead of the manual option.

Install PulledPork pre-requisites.

```
sudo apt-get install libcrypt-ssleay-perl liblwp-
   ↪ useragent-determined-perl
```

Download latest version of PulledPork.

```
cd ~/snort_src
```

```
wget https://github.com/shirkdog/pulledpork/archive/
   ↪ master.tar.gz -O pulledpork-master.tar.gz
```

Extract PulledPork and copy to snort directory.

```
tar xzvf pulledpork-master.tar.gz
cd pulledpork-master/
sudo cp pulledpork.pl /usr/local/bin
sudo chmod +x /usr/local/bin/pulledpork.pl
sudo cp etc/*.conf /etc/snort
```

Test PulledPork and check version.

```
/usr/local/bin/pulledpork.pl -V
```

Edit PulledPork configuration file to setup.

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

Replace all <oinkcode> with actual code.

Line 19	replace	<oinkcode>
Line 29	uncomment	for emerging threats ruleset
Line 74	edit	rule_path=/etc/snort/rules/snort.rules
Line 89	edit	local_rules=/etc/snort/rules/local.rules
Line 92	edit	sid_msg=/etc/snort/sid-msg.map
Line 96	edit	sid_msg_version=2
Line 119	edit	config_path=/etc/snort/snort.conf
Line 133	edit	distro=Debian-6-0
Line 141	edit	black_list=/etc/snort/rules/iplists/black_list.rules
Line 150	edit	IPRVersion=/etc/snort/rules/iplists

Run PulledPork manually.

-l	write comprehensive logs to /var/log
-c /etc/snort/snort.conf	configuration file path

```
sudo /usr/local/bin/pulledpork.pl -c /etc/snort/
   ↪ pulledpork.conf -l
```

At this point */etc/snort/rules/* has *snort.rules*. The file includes all the rules in one file.

Edit Snort configuration file to add PulledPork created rules file.

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

Add at Line 550.

```
include $RULE_PATH/snort.rules
```

Test Snort with new loaded rules.

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

Set *crontab* to daily run PulledPork.

```
sudo crontab -e
```

```
31 20 * * * /usr/local/bin/pulledpork.pl -c /etc/snort
   ↪ /pulledpork.conf -l
```

A.2 Snort from PCAP files

Create a new directory to import pre-existing PCAP files to.

```
mkdir ext_pcaps
```

Install rsync.

```
sudo apt-get install rsync
```

Import (copy) PCAP file.

```
rsync -avP user@host:/pathto/pcaps/bigFlows.pcap
   ↪ ext_pcaps/
```

Run Snort with PCAP file.

```
sudo snort --pcap-single=ext_pcaps/bigFlows.pcap -v -c
   ↪  /etc/snort/snort.conf
```

A.3 Startup scripts

Create and edit a Snort service file.

```
sudo nano /lib/systemd/system/snort.service
```

```
[Unit]
Description=Snort NIDS Daemon
After=syslog.target network.target
[Service]
Type=simple
ExecStart=/usr/local/bin/snort -q -u snort -g snort -c
   ↪  /etc/snort/snort.conf -i eth0
[Install]
WantedBy=multi-user.target
```

Set service to be started at boot.

```
sudo systemctl enable snort
```

Start and check service status.

```
sudo systemctl start snort
```

```
systemctl status snort
```

Create and edit a Barnyard2 service file.

```
sudo nano /lib/systemd/system/barnyard2.service
```

```
[Unit]
Description=Barnyard2 Daemon
After=syslog.target network.target
[Service]
Type=simple
ExecStart=/usr/local/bin/barnyard2 -c /etc/snort/
   ↪ barnyard2.conf -d /var/log/snort -f snort.u2 -q -
   ↪ w /var/log/snort/barnyard2.waldo -g snort -u
   ↪ snort -D -a /var/log/snort/archived_logs
[Install]
WantedBy=multi-user.target
```

Note: *-D* to run as a daemon and *-a /var/log/snort/archived_logs* to move/archive processed logs.

Set service to be started at boot.

```
sudo systemctl enable barnyard2
```

Start and check service status.

```
sudo systemctl start barnyard2
```

```
systemctl status barnyard2
```

Troubleshooting

Check db files.

```
sudo ls -lh /var/lib/mysql/snort/
```

Check table status.

```
sudo mysql
```

```
show table status from snort;
```

Delete table's content.

```
sudo mysql
```

```
use snort;
DELETE FROM sensor;
DELETE FROM event;
DELETE FROM iphdr;
DELETE FROM tcphdr;
DELETE FROM udphdr;
DELETE FROM icmphdr;
DELETE FROM data;
DELETE FROM opt;
DELETE FROM signature;
DELETE FROM sig_class;
DELETE FROM sig_reference;
DELETE FROM reference;
DELETE FROM reference_system;
```

Appendix B

OSSIM

B.1 OSSIM basic setup

Install OSSIM from *current_ossim_iso*.

Note: With KVM use SATA, otherwise GRUB won't install.

B.2 Integrate Snort with OSSIM

Note: This section is based on *Integrating Snort-2.9.8.x with AlienVault OSSIM* by William Parker [20].

@Snort

Backup rsyslog.conf.

```
sudo cp /etc/rsyslog.conf /etc/rsyslog.conf.bak
```

Edit *rsyslog.conf* and add after the last ModLoad using OSSIM IP address.

```
sudo nano /etc/rsyslog.conf
```

```
# Added for OSSIM integration with snort
$SystemLogRateLimitInterval 10
$SystemLogRateLimitBurst 500
#$SystemLogSocketFlowControl on
#$AddUnixListenSocket /var/snort/dev/log
local1.info @@xx.xx.xx.xx:514
```

Restart rsyslog.

```
sudo service rsyslog restart
```

Edit Snort configuration file to configure output plugin for syslog.

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

Add at *# Step #6*.

```
output alert_fast: snort.fast
output alert_syslog: LOG_LOCAL1 LOG_INFO
```

@OSSIM

Create and edit *remote-snort-sensors.conf*.

```
nano /etc/rsyslog.d/remote-snort-sensors.conf
```

```
# Remote Snort sensor logging
$ModLoad imtcp
$InputTCPServerRun 514
# do this in FRONT of the local/regular rules
if $fromhost-ip == 'xx.xx.xx.xx' then /var/log/snort/
   ↪ alert
& ~
#if $fromhost-ip == 'xx.xx.xx.xx' then /var/log/snort/
   ↪ alert
#& ~
```

Backup *rsyslog.conf*.

```
cp /etc/rsyslog.conf /etc/rsyslog.conf.bak
```

Edit *rsyslog.conf*.

```
nano /etc/rsyslog.conf
```

```
$SystemLogRateLimitInterval 10
$SystemLogRateLimitBurst 500
```

Restart rsyslog.

```
/etc/init.d/rsyslog restart
```

@Snort

Send test logging message to the OSSIM box (which will show up in */var/log/snort/alert*.

```
logger -p local1.info "Test from Snort1"
```

@OSSIM

Check */var/log/snort/alert*.

```
nano /var/log/snort/alert
```

Get back to the AlienVault Setup interface.

```
alienvault-setup
```

Scroll down to option 1 *Configure Sensor* and hit <Enter>.
Scroll down to option 4 *Configure Data Source Plugins* and hit <Enter>.
Enable *snort_syslog* plugin.
Browse back and select option 8 *Apply all Changes*.

Note: To reduce unintentional alerts remove unnecessary Data Sources. Note: At this stage you should confirm the Snort asset as a sensor in the WebGUI.

Troubleshooting

Catch messages on OSSIM with *tcpdump*.

```
tcpdump -ni eth0 port 514
```

Count messages on OSSIM with *tcpdump*.

```
tcpdump -i eth0 -v -w /dev/null src xx.xx.xx.xx and
   ↪ port 514
```

Restart the Syslog Collector and the AlienVault Agent Service.

```
/etc/init.d/rsyslog restart
/etc/init.d/ossim-agent restart
```

Reinitialize the OSSIM configuration.

```
ossim-reconfig -c -v -d
```

B.3 Integrate Suricata with OSSIM

@Suricata

Note: This configuration liaises on the previous steps at OSSIM.

Backup rsyslog.conf.

```
sudo cp /etc/rsyslog.conf /etc/rsyslog.conf.bak
```

Edit *rsyslog.conf* and add after the last ModLoad using OSSIM IP address.

```
sudo nano /etc/rsyslog.conf
```

```
$ModLoad imfile
$InputFileName /var/log/suricata/fast.log
$InputFileTag suri
$InputFileFacility local3
$InputRunFileMonitor
local3.* @xx.xx.xx.xx:514
```

Restart rsyslog.

```
sudo service rsyslog restart
```

Appendix C

Suricata

C.1 Suricata basic setup

Note: This section is based on https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Debian_Installation.

Install *tcpdump*.

```
sudo apt-get install tcpdump
```

Install *less*.

```
sudo apt-get install less
```

Activate interface on bridge.

```
sudo ip link set eth0 up
```

Set interface to be persistent.

```
sudo nano /etc/network/interfaces
```

```
auto eth0
iface eth0 inet manual
        up ifconfig eth0 up
```

Check activity at the interface.

```
sudo tcpdump -i eth0
```

Create a Suricata directory.

```
mkdir ~/suricata_src
cd ~/suricata_src
```

Install CA certificates for the necessary downloads.
Not advisable alternative using *wget -no-check-certificate*.

```
sudo apt-get install ca-certificates
```

Download latest version of Suricata.

```
wget https://www.openinfosecfoundation.org/download/
   ↪ suricata-4.0.4.tar.gz
```

Install pre-requisites.

```
sudo apt-get install libpcre3 libpcre3-dbg libpcre3-
   ↪ dev build-essential autoconf automake libtool
   ↪ libpcap-dev libnet1-dev libyaml-0-2 libyaml-dev
   ↪ zlib1g zlib1g-dev libmagic-dev libcap-ng-dev
   ↪ libjansson-dev pkg-config
```

Note: We are skipping IPS, otherwise: *apt-get -y install libnetfilter-queue-dev*.

Extract Suricata.

```
tar -xvzf suricata-4.0.4.tar.gz
cd suricata-4.0.4
```

Note: We are skipping IPS, otherwise *./configure –enable-nfqueue –prefix=/usr –sysconfdir=/etc –localstatedir=/var*

Compile and install.

```
./configure --prefix=/usr --sysconfdir=/etc --
   ↪ localstatedir=/var
make
sudo make install
sudo ldconfig
```

Note: This section is based on https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Basic_Setup.

Create the Suricata directories.

```
sudo mkdir /var/log/suricata
sudo mkdir /etc/suricata
```

Copy configuration files and dynamic preprocessors.
classification.config
reference.config
suricata.yaml

```
sudo cp classification.config /etc/suricata
sudo cp reference.config /etc/suricata
sudo cp suricata.yaml /etc/suricata
```

Note: This section is based on https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Snortconf_to_Suricatayaml.

Edit Suricata configuration file.

```
sudo nano /etc/suricata/suricata.yaml
```

Edit *HOME_NET* value (e.g. Line 17).

```
HOME_NET: "[xx.xx.xx.x/xx]"
```

Note: Check relevant section for PF_RING setup.

C.1.1 Oinkmaster

Note: This section is based on https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Rule_Management_with_Oinkmaster.

Install Oinkmaster.

```
sudo apt-get install oinkmaster
```

Edit Oinkmaster configuration file.

```
sudo nano /etc/oinkmaster.conf
```

Add URL e.g. at Line 73.

```
url = http://rules.emergingthreats.net/open/suricata/
   ↪ emerging.rules.tar.gz
```

Create a rules directory.

```
sudo mkdir /etc/suricata/rules
```

Download initial rule set.

```
cd /etc
sudo oinkmaster -C /etc/oinkmaster.conf -o /etc/
   ↪ suricata/rules
```

Edit Suricata configuration file to refer the new rules configuration files.

```
sudo nano /etc/suricata/suricata.yaml
```

Edit at Line 110 and 111.

```
classification-file: /etc/suricata/rules/
   ↪ classification.config
reference-config-file: /etc/suricata/rules/reference.
   ↪ config
```

Test Suricata.

```
sudo suricata -c /etc/suricata/suricata.yaml -i eth0
```

Check available rules.

```
ls /etc/suricata/rules/*.rules
```

Available rules can be added editing *suricata.yaml*.

```
sudo nano /etc/suricata/suricata.yaml
```

A disabled rule will be enabled next time Oinkmaster runs. The reverse is also true. Exceptions can be set in Oinkmaster instead editing *oinkmaster.conf* at the bottom of the file.

```
sudo nano /etc/oinkmaster.conf
```

Rules can also be modified (e.g. for IPS implementation) in the same way.

C.2 Suricata from PCAP files

Create a new directory to import pre-existing PCAP files to.

```
mkdir ext_pcaps
```

Install rsync.

```
sudo apt-get install rsync
```

Import (copy) PCAP file.

```
rsync -avP user@host:/pathto/pcaps/bigFlows.pcap
   ↪ ext_pcaps/
```

Run Suricata with PCAP file.

```
sudo suricata -c /etc/suricata/suricata.yaml -vv -r
   ↪ ext_pcaps/bigFlows.pcap
```

Note: *-v* or *-vv* to increase verbosity.

C.3 Suricata with PF_RING

Note: This section is based on https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Installation_of_Suricata_stable_with_PF_RING_(STABLE)_on_Ubuntu_server_1204

Install pre-requisites.

```
sudo apt-get install build-essential bison flex linux-
   ↪ headers-$(uname -r) libnuma-dev
```

Download PF_RING.

```
wget https://github.com/ntop/PF_RING/archive/7.0.0.tar
   ↪ .gz
```

Extract, compile and install PF_RING.

```
tar -xvzf 7.0.0.tar.gz
cd PF_RING-7.0.0/
make
cd kernel/
sudo make install
cd ../userland/lib
sudo make install
```

Load and check PF_RING.

```
sudo modprobe pf_ring
modinfo pf_ring && cat /proc/net/pf_ring/info
```

Build Suricata with PF_RING.

```
cd
cd suricata_src/
cd suricata-4.0.4/
./configure --prefix=/usr --sysconfdir=/etc --
   ↪ localstatedir=/var --enable-pfring --with-
   ↪ libpfring-includes=/usr/local/pfring/include --
   ↪ with-libpfring-libraries=/usr/local/pfring/lib
make
sudo make install
sudo ldconfig
```

Appendix D

Pytbull

D.1 Pytbull Client

Add non-free to sources.list.

```
sudo nano /etc/apt/sources.list
```

Install pre-requisites.

```
sudo apt-get install python python-scapy python-
   ↪ feedparser python-cherrypy3
sudo apt-get install nmap hping3 nikto tcpreplay
   ↪ apache2-utils
sudo apt-get install aptitude
sudo aptitude install build-essential checkinstall
   ↪ libssl-dev libssh-dev
```

Install CA certificates for the necessary downloads.
Not advisable alternative using *wget -no-check-certificate*.

```
sudo apt-get install ca-certificates
```

Download ncrack.

```
wget https://nmap.org/ncrack/dist/ncrack-0.5.tar.gz
```

Extract, compile and install ncrack.

```
tar -xzf ncrack-0.5.tar.gz
cd ncrack-0.5
./configure
make
sudo make install
```

Download pytbull.

```
cd /usr/local/src/
sudo wget https://downloads.sourceforge.net/project/
   ↪ pytbull/pytbull-2.0.tar.bz2
```

Extract pytbull.

```
sudo tar xvf pytbull-2.0.tar.bz2
sudo mv pytbull/ /opt/
cd /opt/pytbull/
```

Update pytbull if available.

```
sudo mv pytbull/ pytbull20
sudo apt-get install mercurial
cd /opt/
sudo hg clone http://pytbull.hg.sourceforge.net:8000/
   ↪ hgroot/pytbull/pytbull
```

D.2 Pytbull Server

Install pre-requisites.

```
sudo apt-get install python
sudo apt-get install vsftpd apache2 openssh-server
```

Appendix E

Snort 3

E.1 Snort 3 basic setup

Note: This section is based on *Snort_3.0.0-a4-241_on_Ubuntu_14_and_16* by Noah Dietrich from https://www.snort.org/documents/

Install dependencies.

```
sudo apt-get install -y build-essential autotools-dev
   ↪ libdumbnet-dev libluajit-5.1-dev libpcap-dev
   ↪ libpcre3-dev zlib1g-dev pkg-config libhwloc-dev
   ↪ cmake
```

Install optional features.

```
sudo apt-get install -y liblzma-dev openssl libssl-dev
   ↪  cpputest libsqlite3-dev uuid-dev
```

Install git.

```
sudo apt-get install -y libtool git autoconf
```

Install DAQ dependencies.

```
sudo apt-get install -y bison flex
```

Note: If inline *sudo apt-get install -y libnetfilter-queue-dev*

Create folder.

```
mkdir ~/snortalpha_src
cd ~/snortalpha_src
```

Install safec for C-library calls.

```
cd ~/snortalpha_src
wget https://downloads.sourceforge.net/project/
   ↪ safeclib/libsafec-10052013.tar.gz
tar -xzvf libsafec-10052013.tar.gz
cd libsafec-10052013
./configure
make
sudo make install
```

Install ragel for Hyperscan.

```
cd ~/snortalpha_src
wget http://www.colm.net/files/ragel/ragel-6.10.tar.gz
tar -xzvf ragel-6.10.tar.gz
cd ragel-6.10
./configure
make
sudo make install
```

Download Boost above 1.58 but don't install.

```
cd ~/snortalpha_src
wget https://dl.bintray.com/boostorg/release/1.65.1/
   ↪ source/boost_1_65_1.tar.gz
tar -xvzf boost_1_65_1.tar.gz
```

Install python.

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

Install Hyperscan 4.6.0 referencing Boost location.

```
cd ~/snortalpha_src
wget https://github.com/intel/hyperscan/archive/v4
   ↪ .6.0.tar.gz
tar -xvzf v4.6.0.tar.gz
mkdir ~/snortalpha_src/hyperscan-4.6.0-build
cd hyperscan-4.6.0-build/
cmake -DCMAKE_INSTALL_PREFIX=/usr/local -DBOOST_ROOT
   ↪ =~/snortalpha_src/boost_1_65_1/ ../hyperscan
   ↪ -4.6.0
make
sudo make install
```

Test Hyperscan.

```
cd ~/snortalpha_src/hyperscan-4.6.0-build/
./bin/unit-hyperscan
```

Install Flatbuffers.

```
cd ~/snortalpha_src
wget https://github.com/google/flatbuffers/archive/
   ↪ master.tar.gz -O flatbuffers-master.tar.gz
tar -xvzf flatbuffers-master.tar.gz
mkdir flatbuffers-build
cd flatbuffers-build
cmake ../flatbuffers-master
make
sudo make install
```

Install DAQ.

```
cd ~/snortalpha_src
wget https://www.snort.org/downloads/snortplus/daq
   ↪ -2.2.2.tar.gz
tar -xvzf daq-2.2.2.tar.gz
cd daq-2.2.2
./configure
make
sudo make install
```

Update shared libraries.

```
sudo ldconfig
```

Clone from Git and install.

```
cd ~/snortalpha_src
git clone git://github.com/snortadmin/snort3.git
cd snort3
./configure_cmake.sh --prefix=/opt/snort --enable-
   ↪ shell --enable-large-pcap
cd build
make
sudo make install
```

Alternative to Git's latest (Build 244).

```
cd ~/snortalpha_src
wget https://github.com/snortadmin/snort3/archive/
   ↪ BUILD_241.tar.gz
tar -xvzf BUILD_241.tar.gz
cd snort3-BUILD_241/
```

Note: If using alternative, configure and install Snort as in previous step.

Test Snort.

```
/opt/snort/bin/snort -V
```

Create link to Snort in /usr/sbin.

```
sudo ln -s /opt/snort/bin/snort /usr/sbin/snort
```

Create environmental variables.

```
export LUA_PATH=/opt/snort/include/snort/lua/\?.lua
   ↪ \;\;
export SNORT_LUA_PATH=/opt/snort/etc/snort
sh -c "echo 'export LUA_PATH=/opt/snort/include/snort/
   ↪ lua/\?.lua\;\;' >> ~/.bashrc"
sh -c "echo 'export SNORT_LUA_PATH=/opt/snort/etc/
   ↪ snort' >> ~/.bashrc"
sudo visudo
        Defaults env_keep += "LUA_PATH SNORT_LUA_PATH"
```

Test Snort.

```
/opt/snort/bin/snort -c /opt/snort/etc/snort/snort.lua
```

Note: locations are based on */opt/snort* because *./configure –prefix=/opt/snort*

Note: PulledPork rules are compatible but we tested with Snort3's specific.

Setup Snort3's rules.

```
cd ~/snortalpha_src/
wget https://www.snort.org/downloads/community/snort3-
   ↪ community-rules.tar.gz
tar -xvzf snort3-community-rules.tar.gz
cd snort3-community-rules
sudo mkdir /opt/snort/etc/snort/rules
sudo cp snort3-community.rules /opt/snort/etc/snort/
   ↪ rules/
sudo cp sid-msg.map /opt/snort/etc/snort/rules/
```

Uncomment Snort3's rules.

```
sudo sed -i '17,$s/^# //' /opt/snort/etc/snort/rules/
   ↪ snort3-community.rules
```

Test community rules.

```
/opt/snort/bin/snort -c /opt/snort/etc/snort/snort.lua
   ↪  -R /opt/snort/etc/snort/rules/snort3-community.
   ↪ rules
```

Enable decoder and inspector.

```
sudo nano /opt/snort/etc/snort/snort.lua
```

Uncomment enable_builtin_rules = true at line 195.

Test built-in rules.

```
/opt/snort/bin/snort -c /opt/snort/etc/snort/snort.lua
```

Test both rules.

```
/opt/snort/bin/snort -c /opt/snort/etc/snort/snort.lua
   ↪  -R /opt/snort/etc/snort/rules/snort3-community.
   ↪ rules
```

Edit for output plugin (line 254, uncomment and edit)

```
sudo nano /opt/snort/etc/snort/snort.lua
```

```
        alert_csv = {file = true,}
```

E.2 Snort 3 from PCAP files

Create folder.

```
sudo mkdir /var/log/snort/
```

Example using PCAPs with *-A alert_fast*.

```
sudo /opt/snort/bin/snort -c /opt/snort/etc/snort/
   ↪ snort.lua --pcap-filter \*.pcap --pcap-dir ~/
   ↪ ext_pcaps_short -R /opt/snort/etc/snort/rules/
   ↪ snort3-community.rules -A alert_fast -s 65535 -k
   ↪ none -q -z 0
```

Custom PCAP example with *-l /var/log/snort*.

```
sudo /opt/snort/bin/snort -c /opt/snort/etc/snort/
   ↪ snort.lua --pcap-filter \*.pcap --pcap-dir ~/
   ↪ ext_pcaps_short -R /opt/snort/etc/snort/rules/
   ↪ snort3-community.rules -l /var/log/snort -s 65535
   ↪  -k none -z 0
```

chmod output at /var/log/snort/alert_csv.txt (after first run).

```
sudo chmod a+r /var/log/snort/alert_csv.txt
```

Count number of alerts.

```
wc -l /var/log/snort/alert_csv.txt
```

Edit snort.lua to reflect PPS in perf_monitor (around line 183).

```
sudo nano /opt/snort/etc/snort/snort.lua
        perf_monitor = { seconds = 1 , cpu = true,
           ↪ modules = {{ name = 'perf_monitor', pegs
           ↪ = [[packets]] }}}
```

Make output to be rw.

```
sudo chmod a+rw /var/log/snort/perf_monitor_base.csv
```

Note: Using *-z 0* results in *0_** prefix when using 2 or more CPUs but not for 1 CPU only.

This page intentionally left blank.

Appendix F

Data

F.1 Allocated CPUs vs PPS

F.1.1 Suricata with long PCAPs

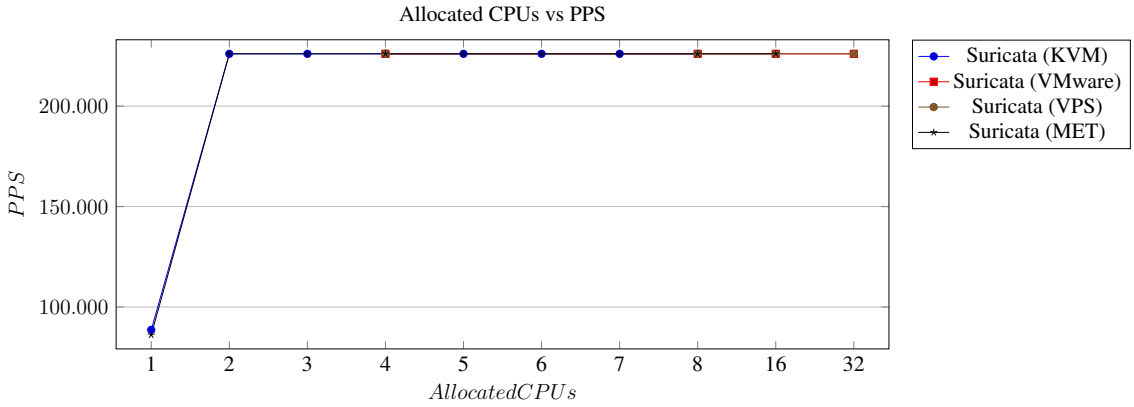

Figure F.1: Suricata: Allocated CPU vs PPS - (bigFlows.pcap)

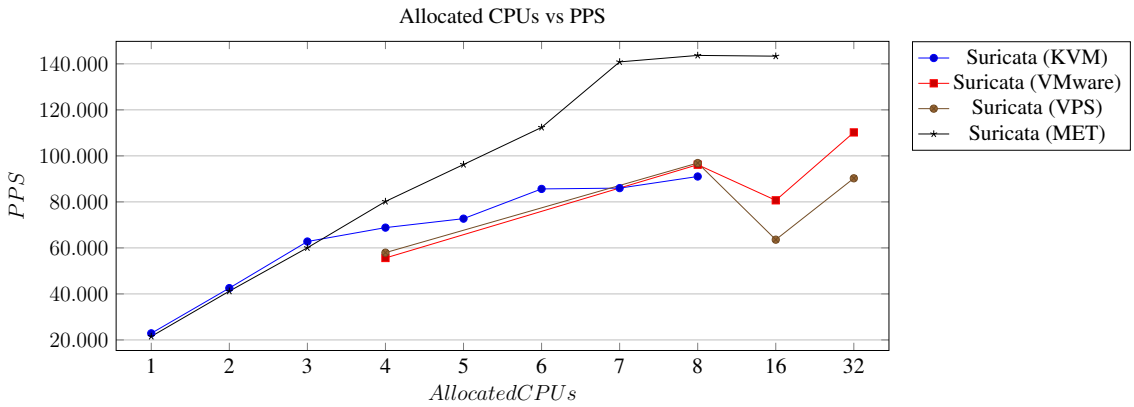

Figure F.2: Suricata: Allocated CPU vs PPS - (maccdc2011 00010 ... .pcap)

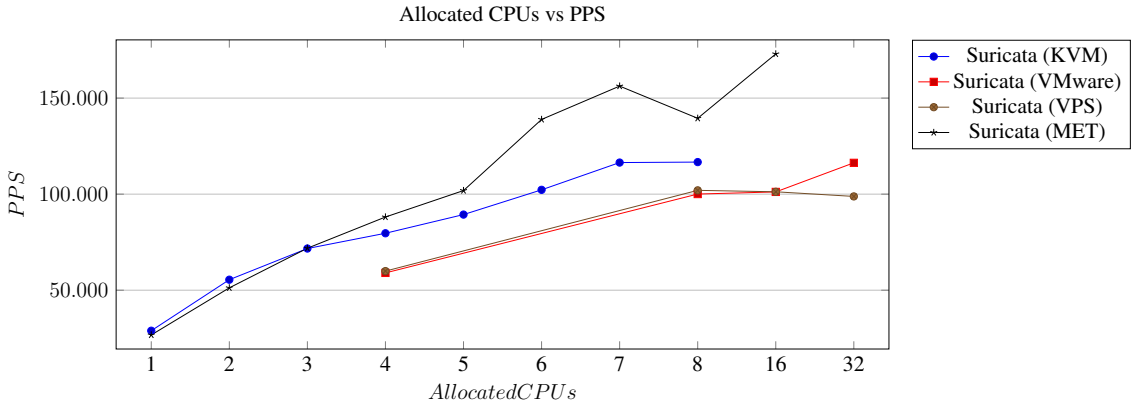

Figure F.3: Suricata: Allocated CPU vs PPS - (maccdc2011 00011 ... .pcap)

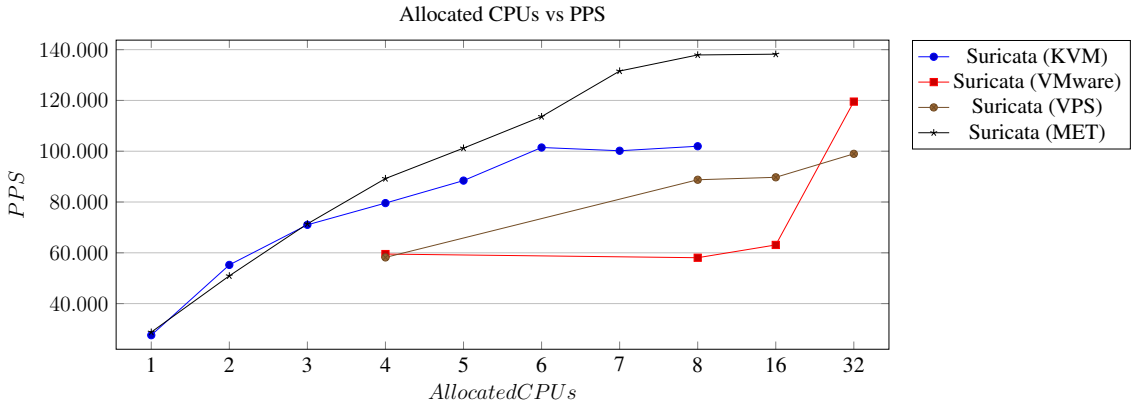

Figure F.4: Suricata: Allocated CPU vs PPS - (maccdc2011 00012 ... .pcap)

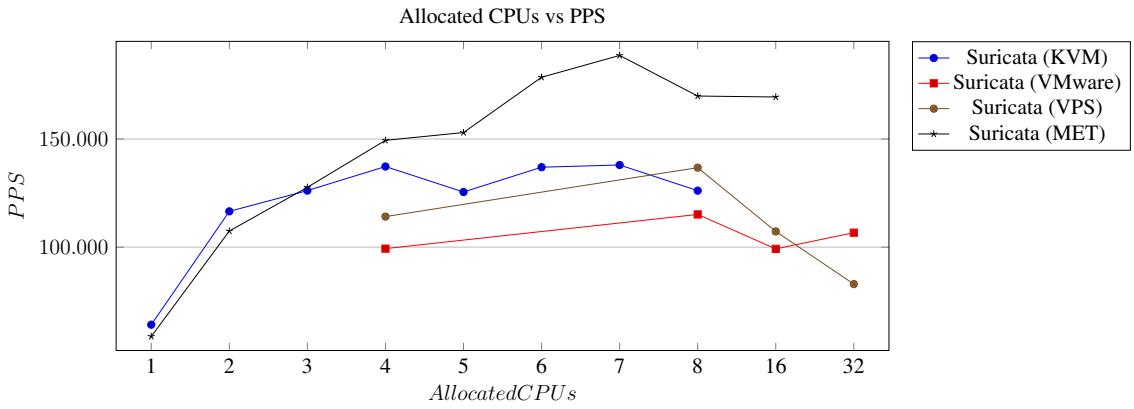

Figure F.5: Suricata: Allocated CPU vs PPS - (maccdc2011 00013 ... .pcap)

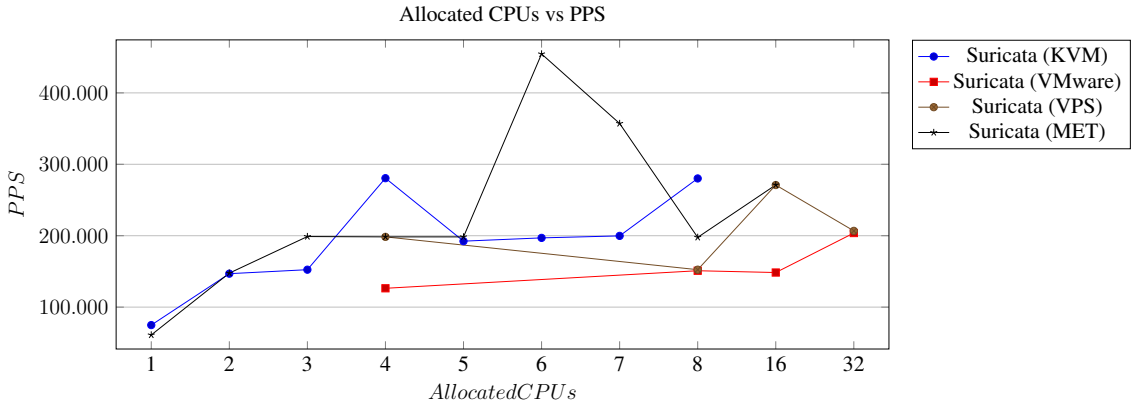

Figure F.6: Suricata: Allocated CPU vs PPS - (maccdc2011 00014 ... .pcap)

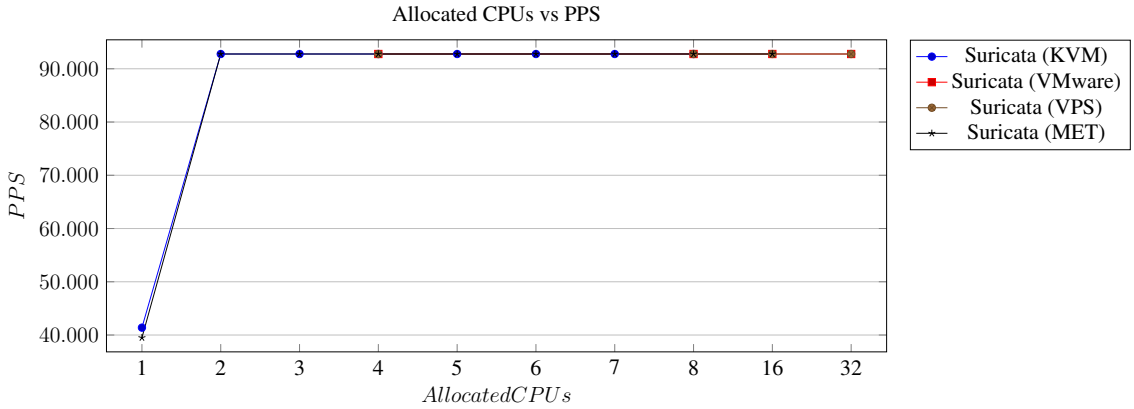

Figure F.7: Suricata: Allocated CPU vs PPS - (purplehaze.pcap)

F.1.2 Snort2 with long PCAPs

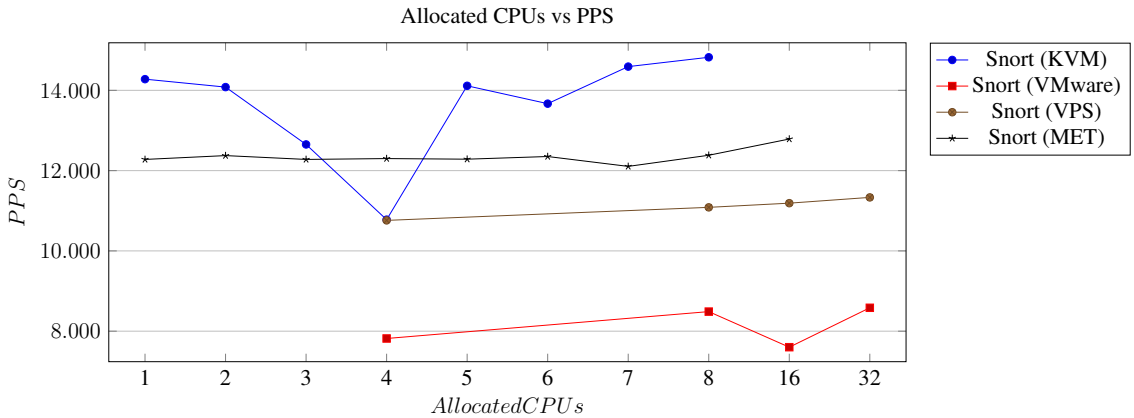

Figure F.8: Snort2: Allocated CPU vs PPS - (bigFlows.pcap)

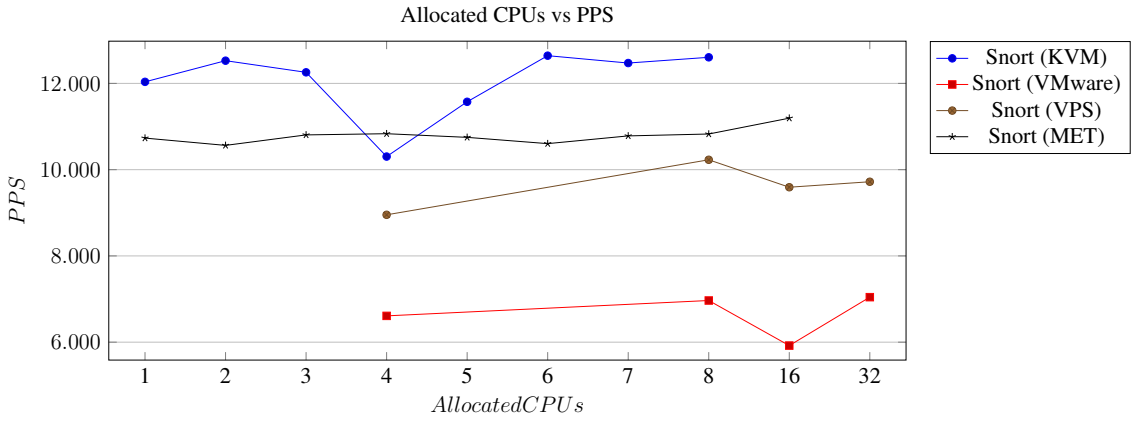

Figure F.9: Snort2: Allocated CPU vs PPS - (maccdc2011 00010 ... .pcap)

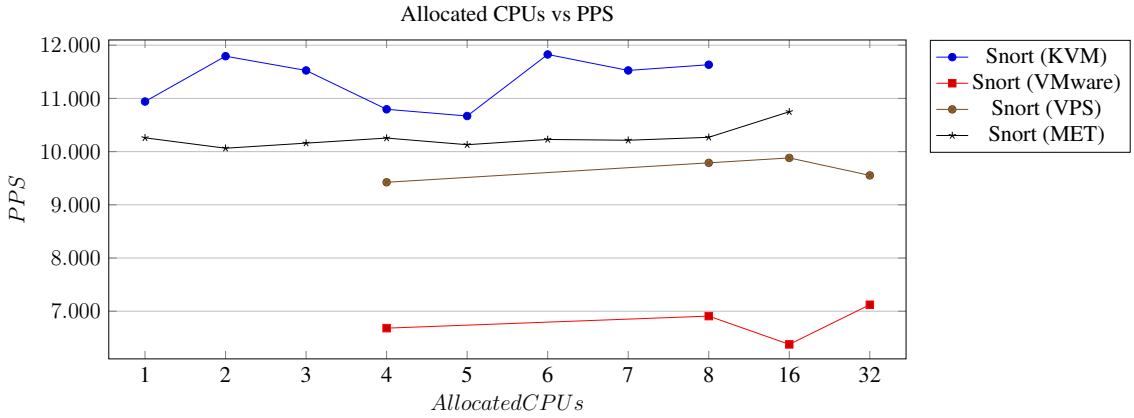

Figure F.10: Snort2: Allocated CPU vs PPS - (maccdc2011 00011 ... .pcap)

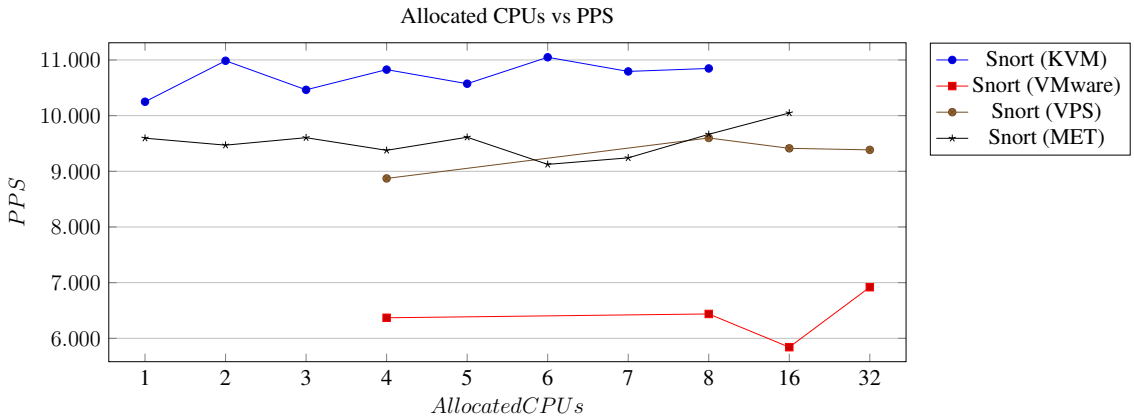

Figure F.11: Snort2: Allocated CPU vs PPS - (maccdc2011 00012 ... .pcap)

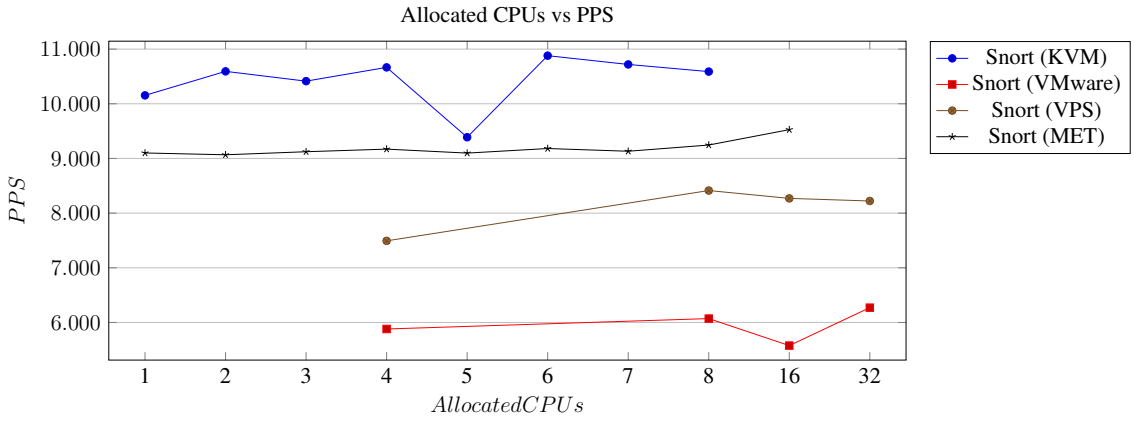

Figure F.12: Snort2: Allocated CPU vs PPS - (maccdc2011 00013 ... .pcap)

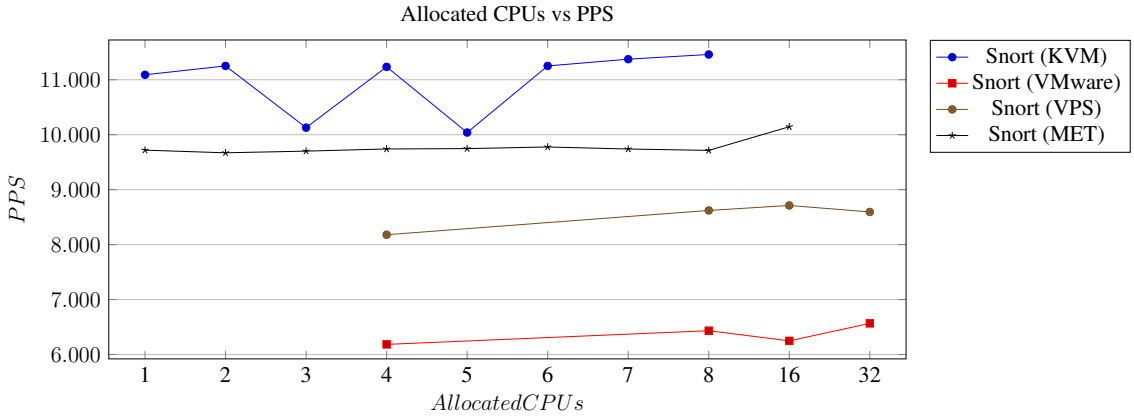

Figure F.13: Snort2: Allocated CPU vs PPS - (maccdc2011 00014 ... .pcap)

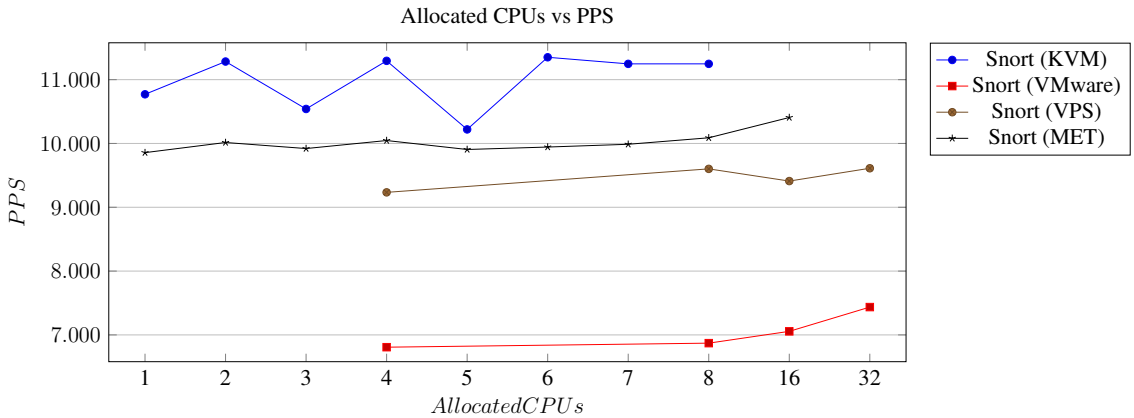

Figure F.14: Snort2: Allocated CPU vs PPS - (purplehaze.pcap)

F.1.3 Snort3 with long PCAPs

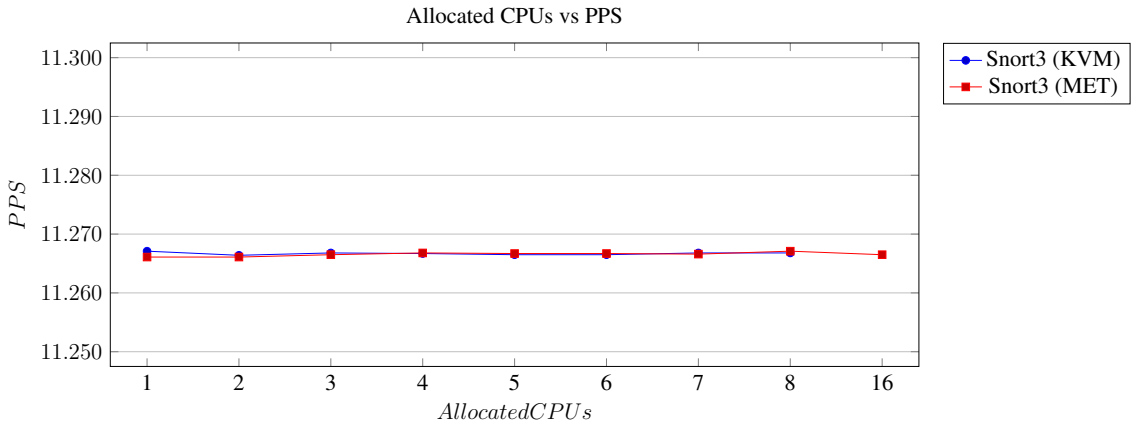

Figure F.15: Snort3: Allocated CPU vs PPS - (bigFlows.pcap)

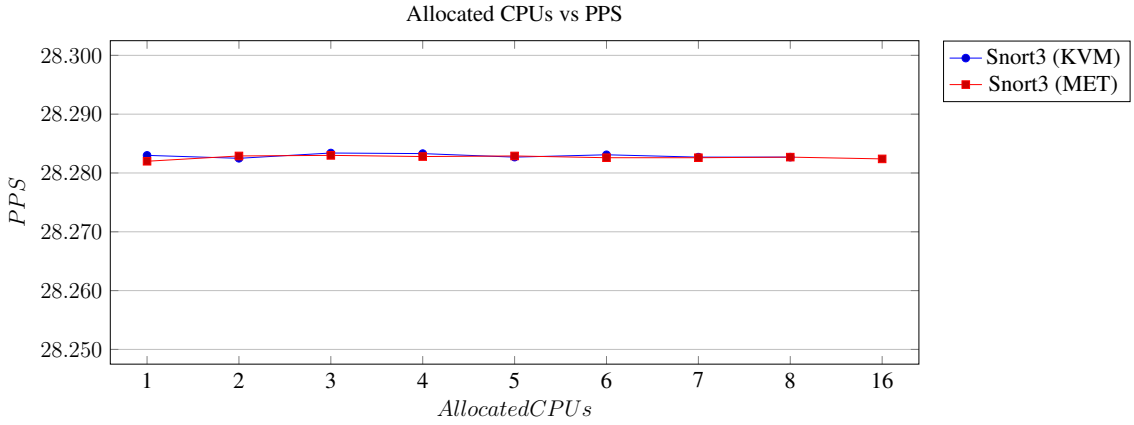

Figure F.16: Snort3: Allocated CPU vs PPS - (maccdc2011 00010 ... .pcap)

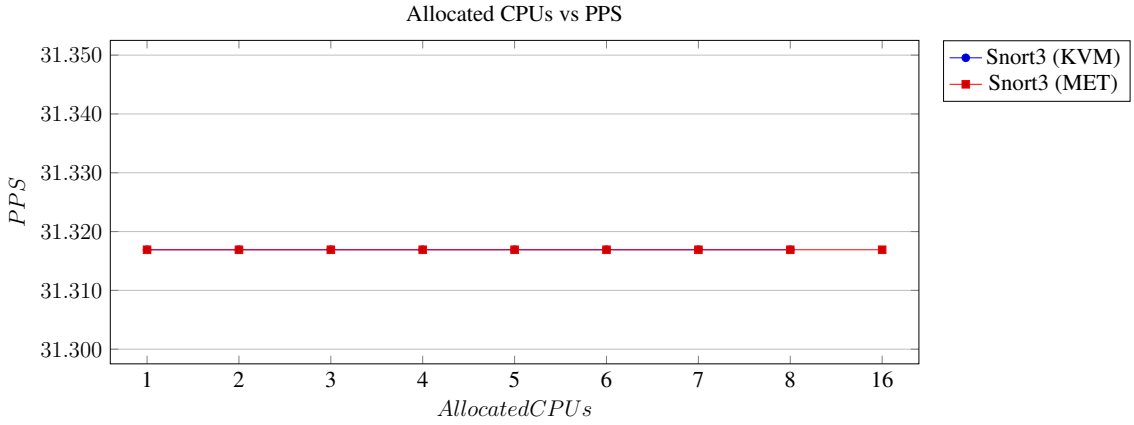

Figure F.17: Snort3: Allocated CPU vs PPS - (maccdc2011 00011 ... .pcap)

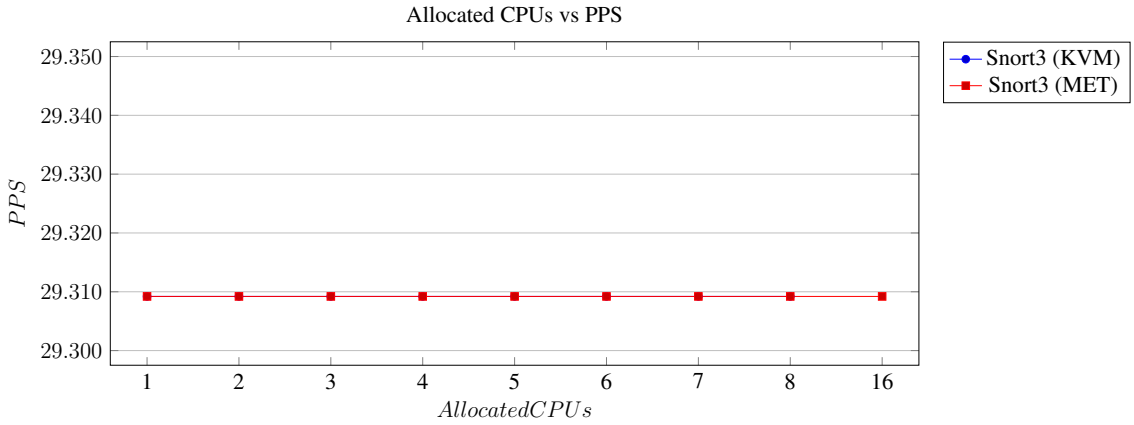

Figure F.18: Snort3: Allocated CPU vs PPS - (maccdc2011 00012 ... .pcap)

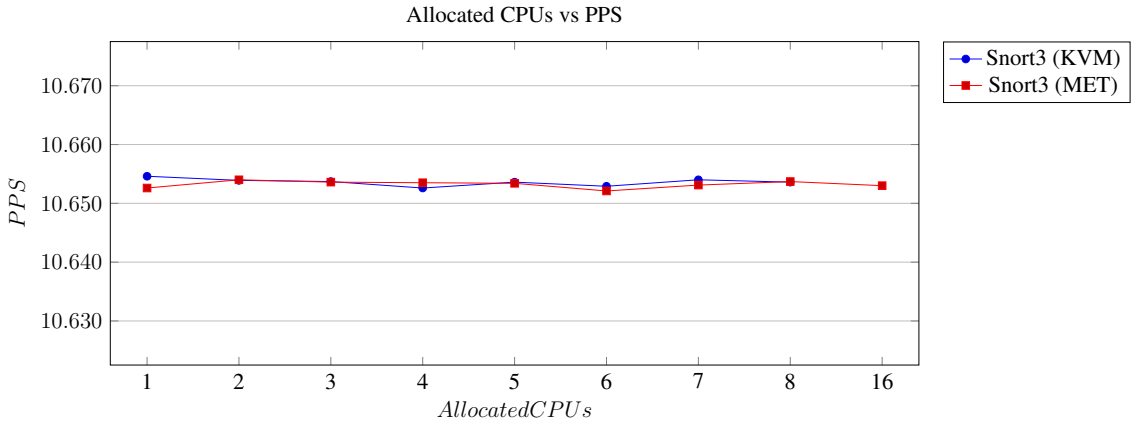

Figure F.19: Snort3: Allocated CPU vs PPS - (maccdc2011 00013 ... .pcap)

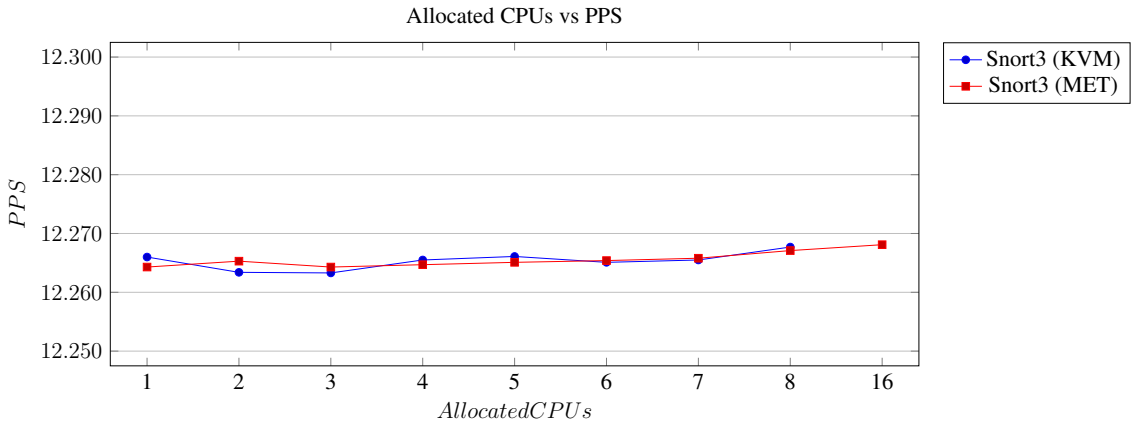

Figure F.20: Snort3: Allocated CPU vs PPS - (maccdc2011 00014 ... .pcap)

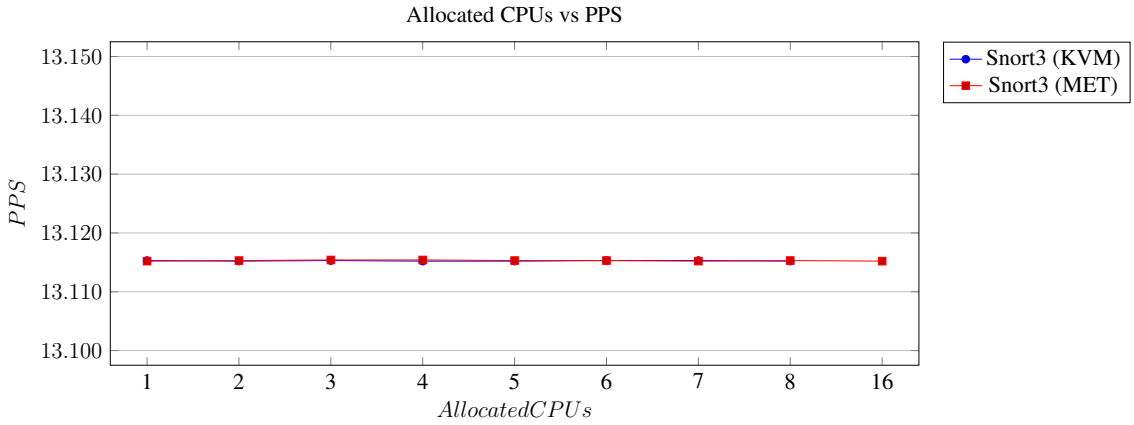

Figure F.21: Snort3: Allocated CPU vs PPS - (purplehaze.pcap)

F.2 Allocated CPUs vs RSS

F.2.1 Suricata with long PCAPs

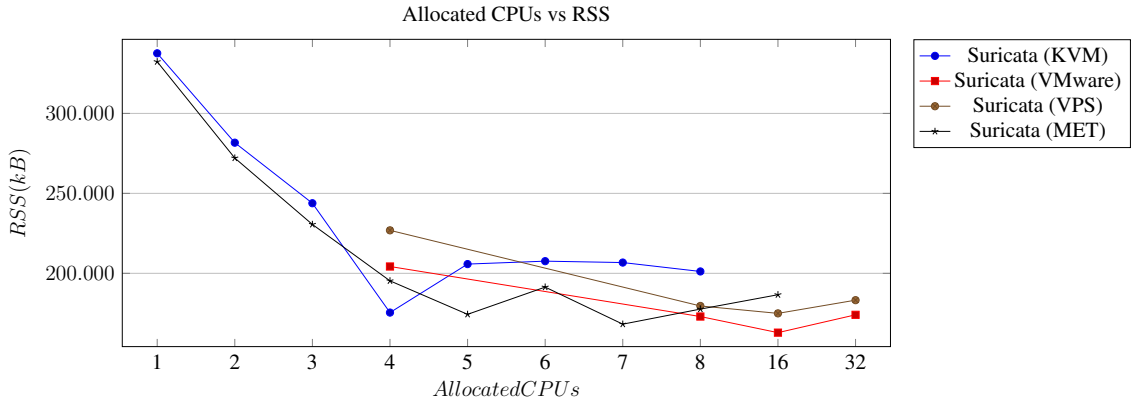

Figure F.22: Suricata: Allocated CPU vs RSS - (bigFlows.pcap)

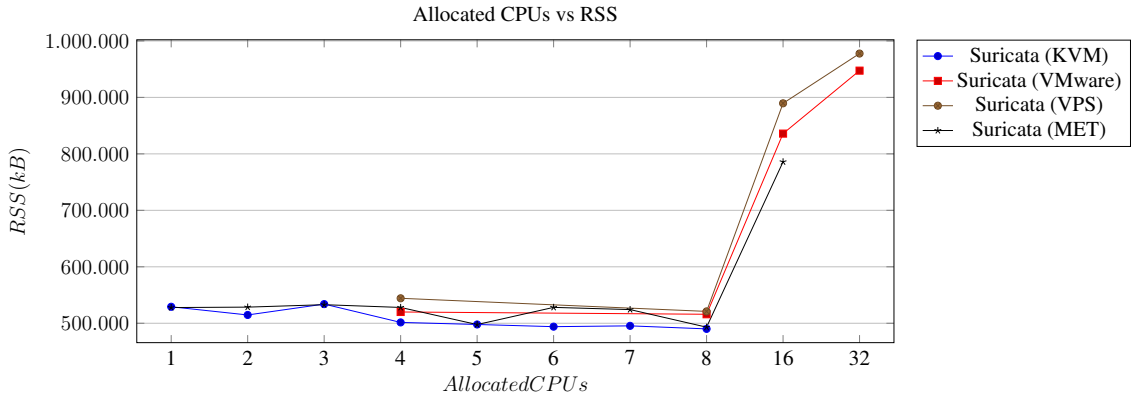

Figure F.23: Suricata: Allocated CPU vs RSS - (maccdc2011 00010 ... .pcap)

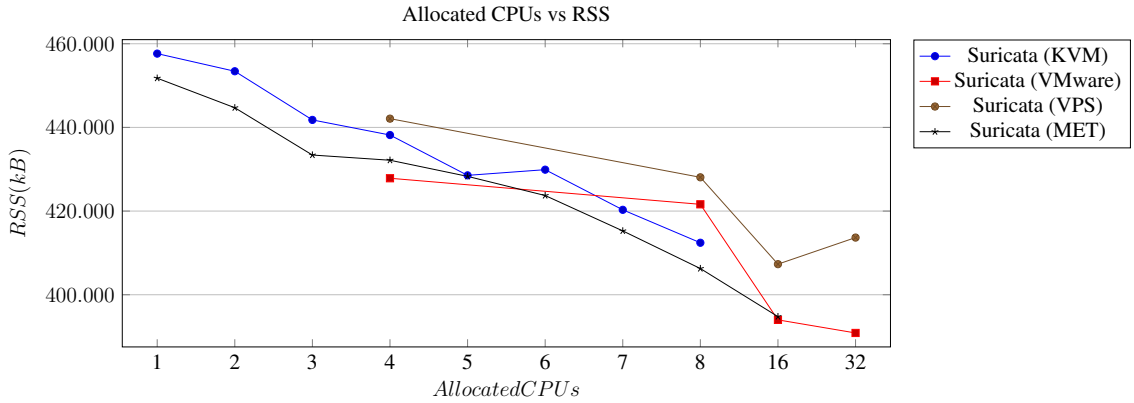

Figure F.24: Suricata: Allocated CPU vs RSS - (maccdc2011 00011 ... .pcap)

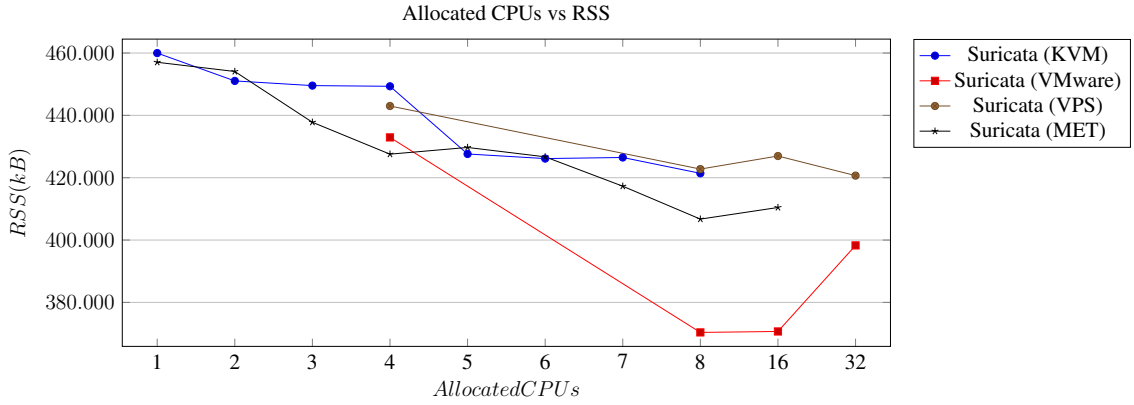

Figure F.25: Suricata: Allocated CPU vs RSS - (maccdc2011 00012 ... .pcap)

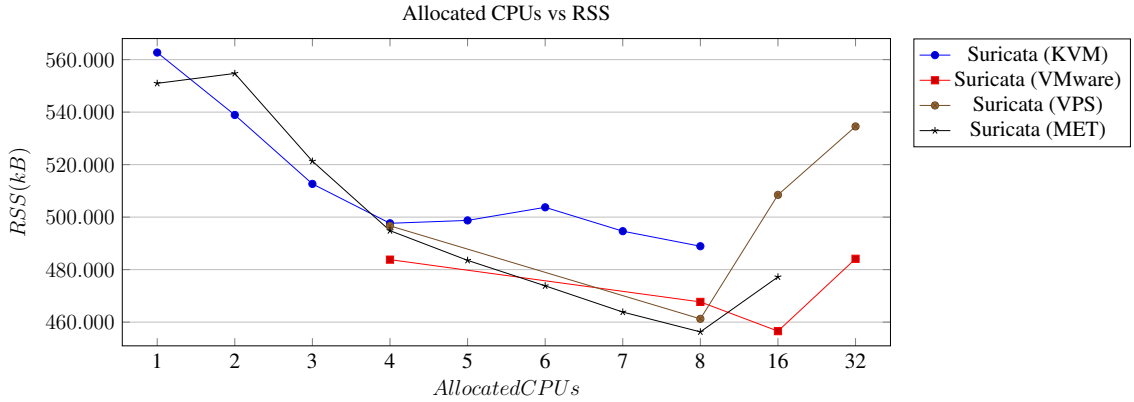

Figure F.26: Suricata: Allocated CPU vs RSS - (maccdc2011 00013 ... .pcap)

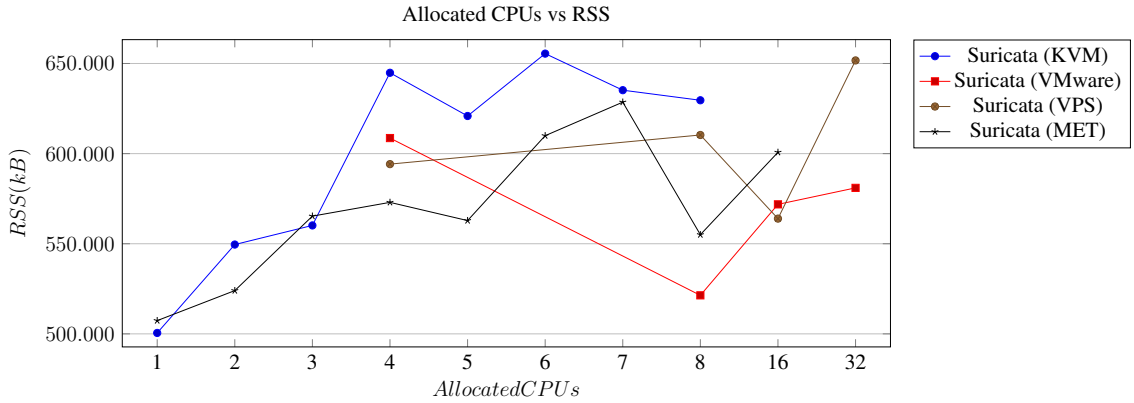

Figure F.27: Suricata: Allocated CPU vs RSS - (maccdc2011 00014 ... .pcap)

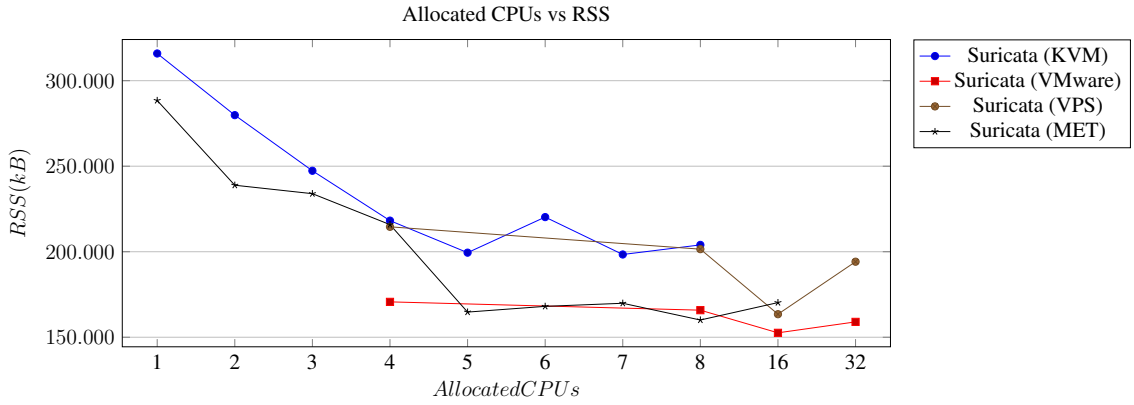

Figure F.28: Suricata: Allocated CPU vs RSS - (purplehaze.pcap)

F.2.2 Snort2 with long PCAPs

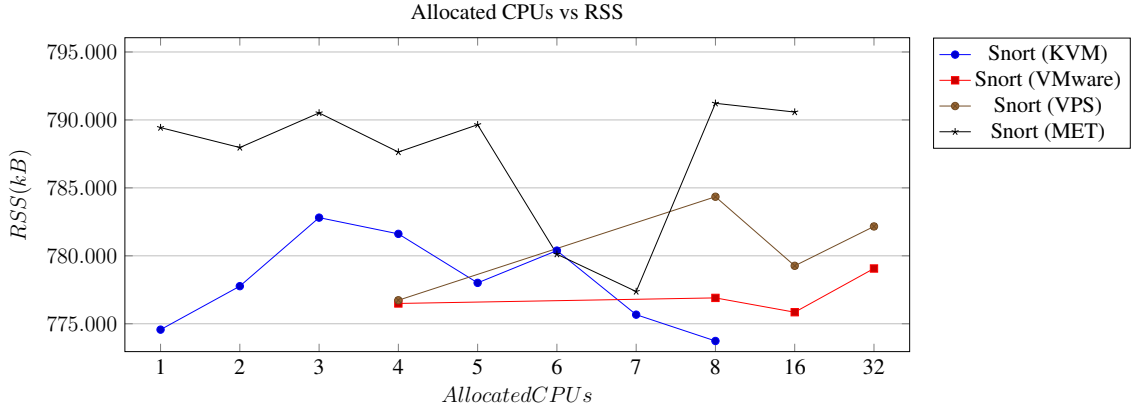

Figure F.29: Snort2: Allocated CPU vs RSS - (bigFlows.pcap)

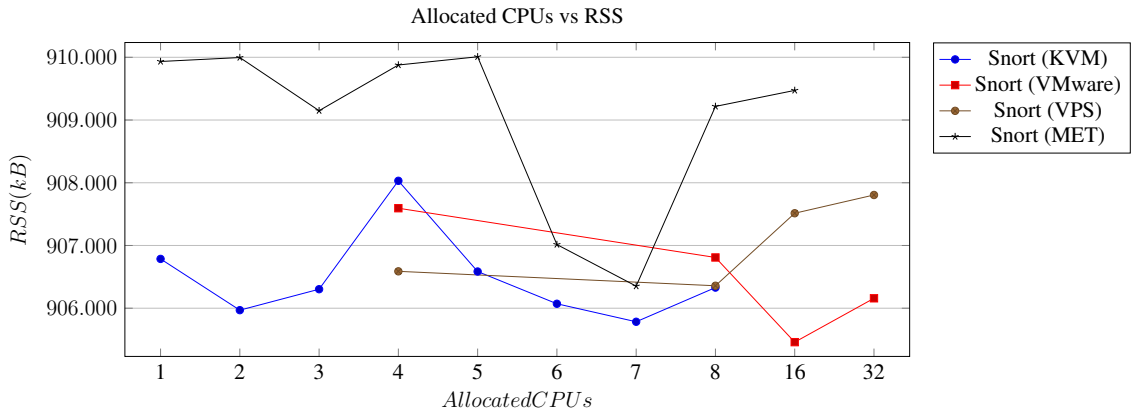

Figure F.30: Snort2: Allocated CPU vs RSS - (maccdc2011 00010 ... .pcap)

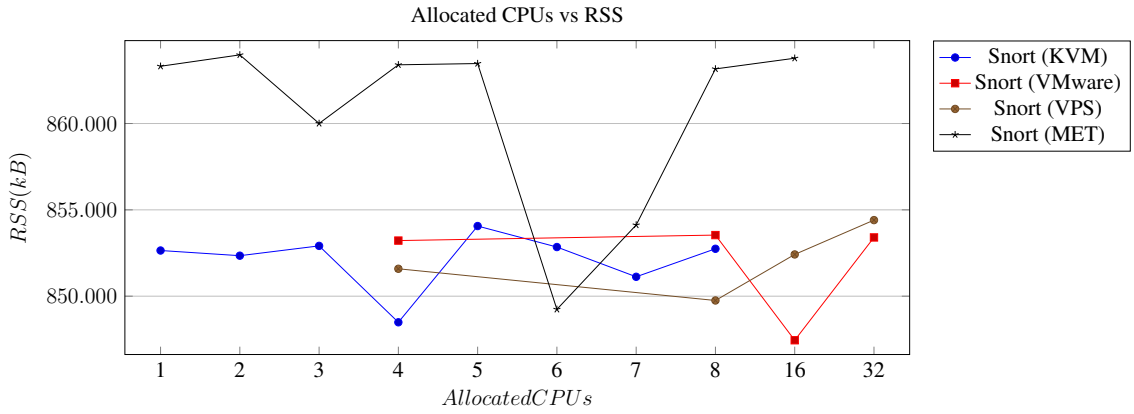

Figure F.31: Snort2: Allocated CPU vs RSS - (maccdc2011 00011 ... .pcap)

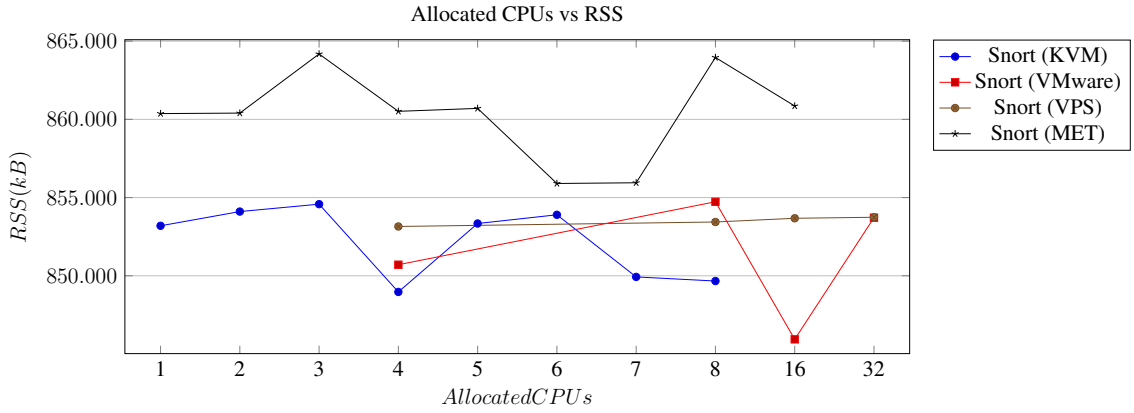

Figure F.32: Snort2: Allocated CPU vs RSS - (maccdc2011 00012 ... .pcap)

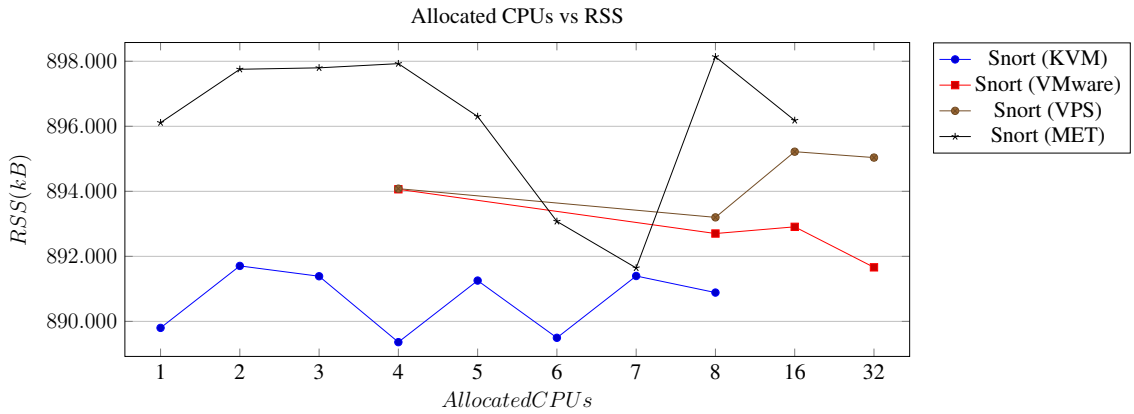

Figure F.33: Snort2: Allocated CPU vs RSS - (maccdc2011 00013 ... .pcap)

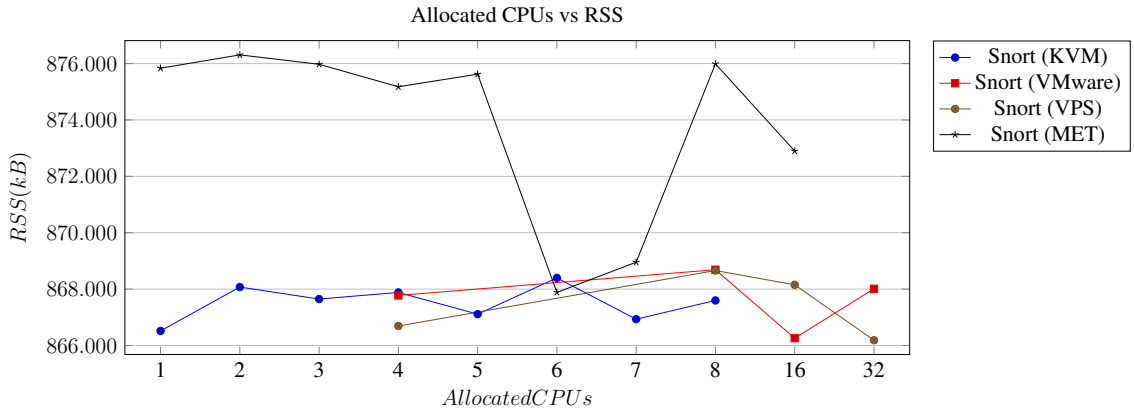

Figure F.34: Snort2: Allocated CPU vs RSS - (maccdc2011 00014 ... .pcap)

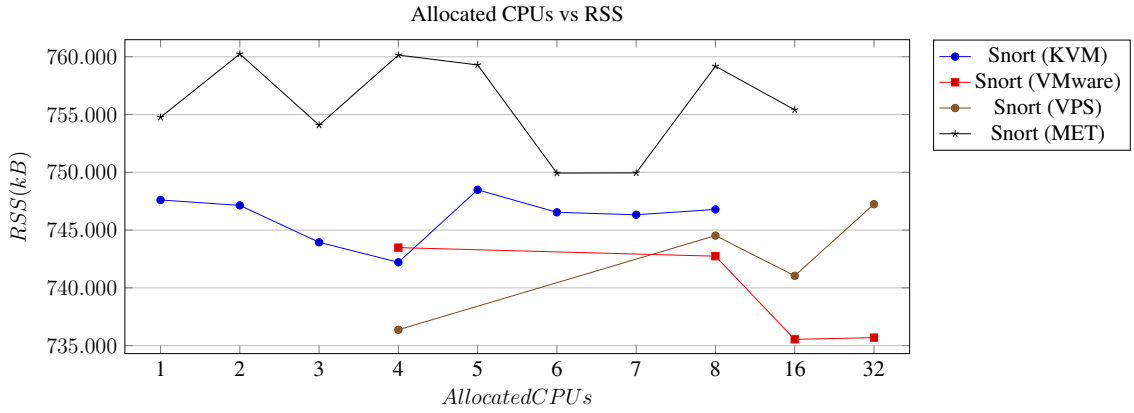

Figure F.35: Snort2: Allocated CPU vs RSS - (purplehaze.pcap)

F.2.3 Snort3 with long PCAPs

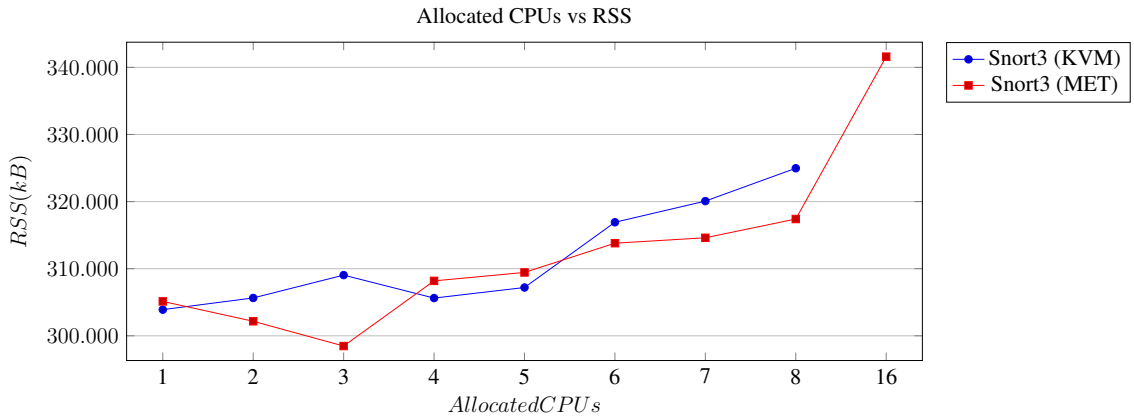

Figure F.36: Snort3: Allocated CPU vs RSS - (bigFlows.pcap)

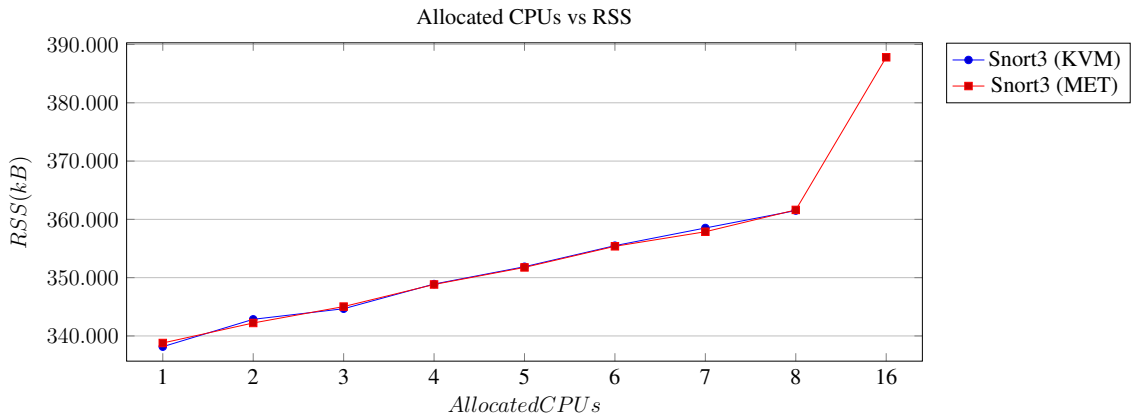

Figure F.37: Snort3: Allocated CPU vs RSS - (maccdc2011 00010 ... .pcap)

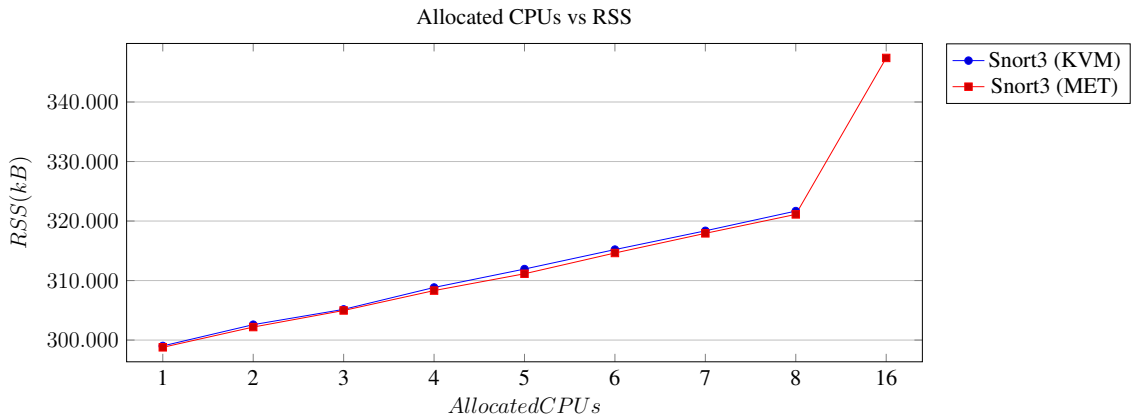

Figure F.38: Snort3: Allocated CPU vs RSS - (maccdc2011 00011 ... .pcap)

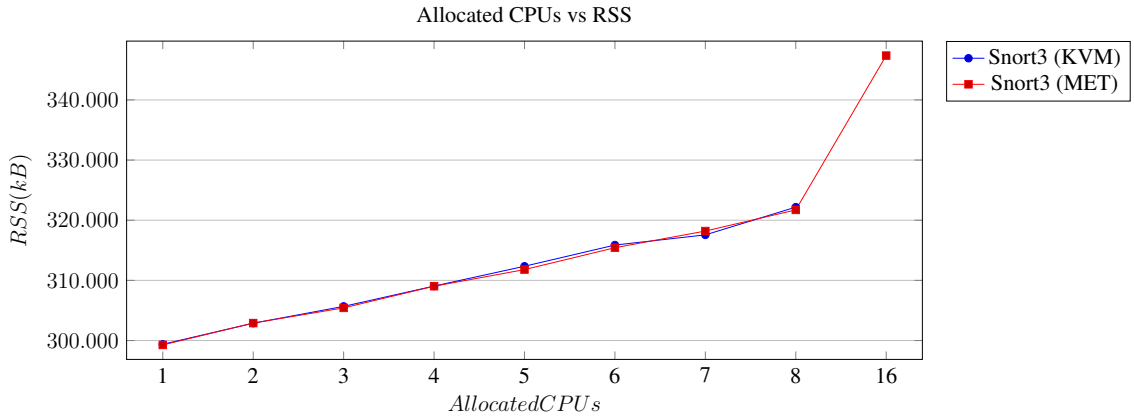

Figure F.39: Snort3: Allocated CPU vs RSS - (maccdc2011 00012 ... .pcap)

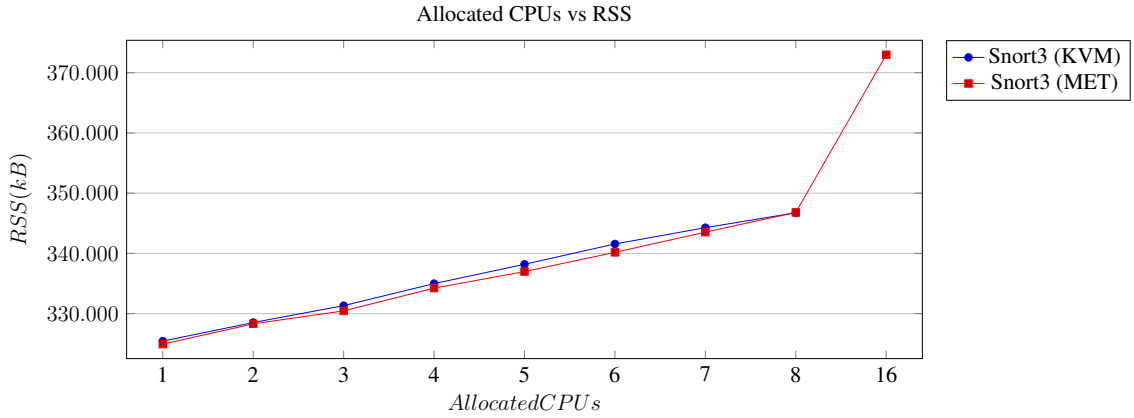

Figure F.40: Snort3: Allocated CPU vs RSS - (maccdc2011 00013 ... .pcap)

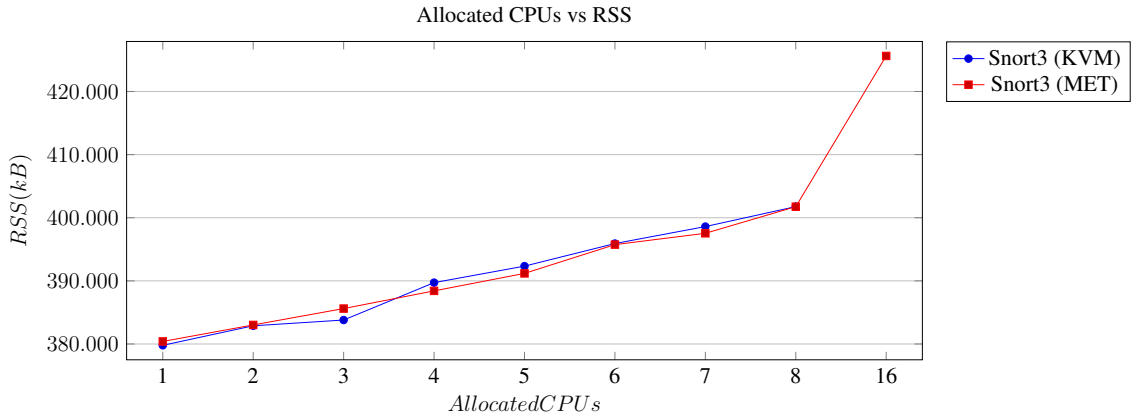

Figure F.41: Snort3: Allocated CPU vs RSS - (maccdc2011 00014 ... .pcap)

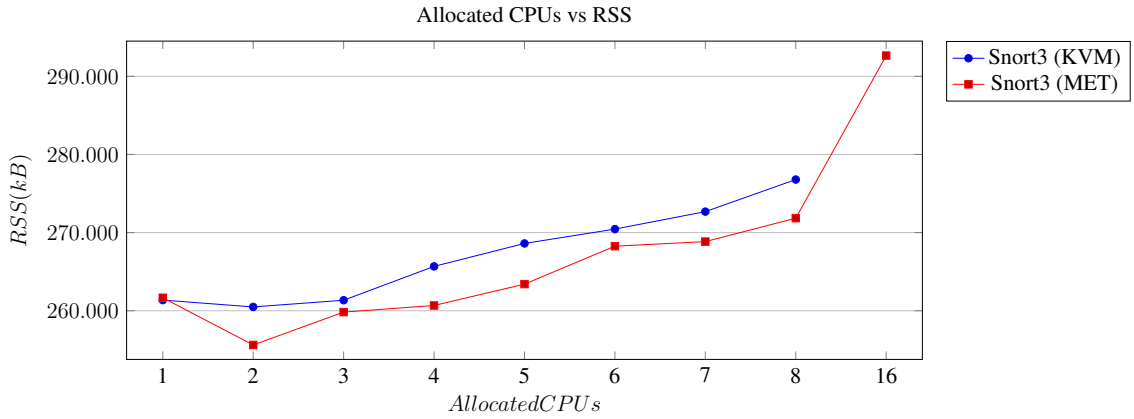

Figure F.42: Snort3: Allocated CPU vs RSS - (purplehaze.pcap)

This page intentionally left blank.

Appendix G

CLI Outputs

G.1 Suricata CLI Outputs

G.1.1 Suricata +8days session

Listing G.1: Suricata with PF_RING at eth0

```
root@host:~# suricata -c /etc/suricata/suricataf0.yaml -vvv --pfring-int=eth0 --pfring-
    ↪ cluster-id=98 --pfring-cluster-type=cluster_round_robin
6/MM/YYYY -- 03:32:37 - <Notice> - This is Suricata version 4.0.4 RELEASE
6/MM/YYYY -- 03:32:37 - <Info> - CPUs/cores online: 16
6/MM/YYYY -- 03:32:37 - <Config> - 'default' server has 'request-body-minimal-inspect-
    ↪ size' columnsset to 33111 and 'request-body-inspect-window' columnsset to 4147
    ↪ after randomization.
6/MM/YYYY -- 03:32:37 - <Config> - 'default' server has 'response-body-minimal-inspect-
    ↪ size' columnsset to 40787 and 'response-body-inspect-window' columnsset to 16306
    ↪ after randomization.
6/MM/YYYY -- 03:32:37 - <Config> - DNS request flood protection level: 500
6/MM/YYYY -- 03:32:37 - <Config> - DNS per flow memcap (state-memcap): 524288
6/MM/YYYY -- 03:32:37 - <Config> - DNS global memcap: 16777216
6/MM/YYYY -- 03:32:37 - <Config> - Protocol detection and parser disabled columnsfor
    ↪ modbus protocol.
6/MM/YYYY -- 03:32:37 - <Config> - Protocol detection and parser disabled columnsfor
    ↪ enip protocol.
6/MM/YYYY -- 03:32:37 - <Config> - Protocol detection and parser disabled columnsfor
    ↪ DNP3.
6/MM/YYYY -- 03:32:37 - <Info> - Found an MTU of 1500 columnsfor 'eth0'
6/MM/YYYY -- 03:32:37 - <Info> - Found an MTU of 1500 columnsfor 'eth0'
6/MM/YYYY -- 03:32:37 - <Config> - allocated 262144 bytes of memory columnsfor the host
    ↪ columnshash... 4096 buckets of size 64
6/MM/YYYY -- 03:32:37 - <Config> - preallocated 1000 hosts of size 136
6/MM/YYYY -- 03:32:37 - <Config> - host memory usage: 398144 bytes, maximum: 33554432
6/MM/YYYY -- 03:32:37 - <Config> - Core dump size columnsset to unlimited.
6/MM/YYYY -- 03:32:37 - <Config> - allocated 3670016 bytes of memory columnsfor the
    ↪ defrag columnshash... 65536 buckets of size 56
6/MM/YYYY -- 03:32:37 - <Config> - preallocated 65535 defrag trackers of size 168
6/MM/YYYY -- 03:32:37 - <Config> - defrag memory usage: 14679896 bytes, maximum:
    ↪ 33554432
6/MM/YYYY -- 03:32:37 - <Config> - stream "prealloc-sessions": 2048 (per thread)
6/MM/YYYY -- 03:32:37 - <Config> - stream "memcap": 67108864
6/MM/YYYY -- 03:32:37 - <Config> - stream "midstream" session pickups: disabled
6/MM/YYYY -- 03:32:37 - <Config> - stream "async-oneside": disabled
6/MM/YYYY -- 03:32:37 - <Config> - stream "checksum-validation": enabled
6/MM/YYYY -- 03:32:37 - <Config> - stream."inline": disabled
6/MM/YYYY -- 03:32:37 - <Config> - stream "bypass": disabled
6/MM/YYYY -- 03:32:37 - <Config> - stream "max-synack-queued": 5
6/MM/YYYY -- 03:32:37 - <Config> - stream.reassembly "memcap": 268435456
6/MM/YYYY -- 03:32:37 - <Config> - stream.reassembly "depth": 1048576
6/MM/YYYY -- 03:32:37 - <Config> - stream.reassembly "toserver-chunk-size": 2629
```

```
32 6/MM/YYYY -- 03:32:37 - <Config> - stream.reassembly "toclient-chunk-size": 2444
33 6/MM/YYYY -- 03:32:37 - <Config> - stream.reassembly.raw: enabled
34 6/MM/YYYY -- 03:32:37 - <Config> - stream.reassembly "segment-prealloc": 2048
35 6/MM/YYYY -- 03:32:37 - <Config> - Delayed detect disabled
36 6/MM/YYYY -- 03:32:37 - <Info> - Running columnsin live mode, activating unix socket
37 6/MM/YYYY -- 03:32:37 - <Config> - pattern matchers: MPM: ac, SPM: bm
38 6/MM/YYYY -- 03:32:37 - <Config> - grouping: tcp-whitelist (default) 53, 80, 139, 443,
       ↪ 445, 1433, 3306, 3389, 6666, 6667, 8080
39 6/MM/YYYY -- 03:32:37 - <Config> - grouping: udp-whitelist (default) 53, 135, 5060
40 6/MM/YYYY -- 03:32:37 - <Config> - prefilter engines: MPM
41 6/MM/YYYY -- 03:32:37 - <Config> - IP reputation disabled
42 6/MM/YYYY -- 03:32:37 - <Config> - Loading rule file: /etc/suricata/rules/botcc.rules
43 6/MM/YYYY -- 03:32:37 - <Config> - Loading rule file: /etc/suricata/rules/ciarmy.rules
44 6/MM/YYYY -- 03:32:37 - <Config> - Loading rule file: /etc/suricata/rules/compromised.
       ↪ rules
45 6/MM/YYYY -- 03:32:37 - <Config> - Loading rule file: /etc/suricata/rules/drop.rules
46 6/MM/YYYY -- 03:32:37 - <Config> - Loading rule file: /etc/suricata/rules/dshield.rules
47 6/MM/YYYY -- 03:32:37 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
       ↪ attack_response.rules
48 6/MM/YYYY -- 03:32:37 - <Config> - Loading rule file: /etc/suricata/rules/emerging-chat.
       ↪ rules
49 6/MM/YYYY -- 03:32:37 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
       ↪ current_events.rules
50 6/MM/YYYY -- 03:32:37 - <Config> - Loading rule file: /etc/suricata/rules/emerging-dns.
       ↪ rules
51 6/MM/YYYY -- 03:32:37 - <Config> - Loading rule file: /etc/suricata/rules/emerging-dos.
       ↪ rules
52 6/MM/YYYY -- 03:32:37 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
       ↪ exploit.rules
53 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-ftp.
       ↪ rules
54 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-imap.
       ↪ rules
55 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
       ↪ malware.rules
56 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-misc.
       ↪ rules
57 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
       ↪ mobile_malware.rules
58 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
       ↪ netbios.rules
59 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-p2p.
       ↪ rules
60 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
       ↪ policy.rules
61 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-pop3.
       ↪ rules
62 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-rpc.
       ↪ rules
63 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-scan.
       ↪ rules
64 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-smtp.
       ↪ rules
65 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-snmp.
       ↪ rules
66 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-sql.
       ↪ rules
67 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
       ↪ telnet.rules
68 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-tftp.
       ↪ rules
69 6/MM/YYYY -- 03:32:38 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
       ↪ trojan.rules
70 6/MM/YYYY -- 03:32:39 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
       ↪ user_agents.rules
71 6/MM/YYYY -- 03:32:39 - <Config> - Loading rule file: /etc/suricata/rules/emerging-voip.
       ↪ rules
72 6/MM/YYYY -- 03:32:39 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
       ↪ web_client.rules
```

```
6/MM/YYYY -- 03:32:39 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
    ↪ web_server.rules
6/MM/YYYY -- 03:32:40 - <Config> - Loading rule file: /etc/suricata/rules/emerging-worm.
    ↪ rules
6/MM/YYYY -- 03:32:40 - <Config> - Loading rule file: /etc/suricata/rules/tor.rules
6/MM/YYYY -- 03:32:40 - <Warning> - [ERRCODE: SC_ERR_NO_RULES(42)] - No rule files match
    ↪  the pattern /etc/suricata/rules/http-events.rules
6/MM/YYYY -- 03:32:40 - <Config> - No rules loaded from http-events.rules.
6/MM/YYYY -- 03:32:40 - <Warning> - [ERRCODE: SC_ERR_NO_RULES(42)] - No rule files match
    ↪  the pattern /etc/suricata/rules/smtp-events.rules
6/MM/YYYY -- 03:32:40 - <Config> - No rules loaded from smtp-events.rules.
6/MM/YYYY -- 03:32:40 - <Warning> - [ERRCODE: SC_ERR_NO_RULES(42)] - No rule files match
    ↪  the pattern /etc/suricata/rules/dns-events.rules
6/MM/YYYY -- 03:32:40 - <Config> - No rules loaded from dns-events.rules.
6/MM/YYYY -- 03:32:40 - <Warning> - [ERRCODE: SC_ERR_NO_RULES(42)] - No rule files match
    ↪  the pattern /etc/suricata/rules/tls-events.rules
6/MM/YYYY -- 03:32:40 - <Config> - No rules loaded from tls-events.rules.
6/MM/YYYY -- 03:32:40 - <Info> - 38 rule files processed. 12529 rules successfully
    ↪ loaded, 0 rules failed
6/MM/YYYY -- 03:32:40 - <Warning> - [ERRCODE: SC_ERR_FOPEN(44)] - Error opening file: "/
    ↪ etc/suricata//threshold.config": No such file or directory
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor tcp-packet
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor tcp-stream
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor udp-packet
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor other-ip
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_uri
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_request_line
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_client_body
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_response_line
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_header
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_header
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_header_names
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_header_names
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_accept
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_accept_enc
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_accept_lang
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_referer
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_connection
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_content_len
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_content_len
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_content_type
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_content_type
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_protocol
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_protocol
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_start
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_start
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_raw_header
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_raw_header
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_method
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_cookie
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_cookie
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_raw_uri
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_user_agent
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_host
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_raw_host
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_stat_msg
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor http_stat_code
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor dns_query
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor tls_sni
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor tls_cert_issuer
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor tls_cert_subject
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor tls_cert_serial
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor dce_stub_data
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor dce_stub_data
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor ssh_protocol
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor ssh_protocol
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor ssh_software
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor ssh_software
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor file_data
6/MM/YYYY -- 03:32:40 - <Perf> - using unique mpm ctx' columnsfor file_data
```

```
6/MM/YYYY -- 03:32:40 - <Info> - 12534 signatures processed. 1158 are IP-only rules, 5309 are inspecting packet payload, 7656 inspect application layer, 0 are decoder event only
6/MM/YYYY -- 03:32:40 - <Config> - building signature grouping structure, stage 1: preprocessing rules... columnscomplete
6/MM/YYYY -- 03:32:40 - <Perf> - TCP toserver: 41 port groups, 40 unique SGH's, 1 copies
6/MM/YYYY -- 03:32:40 - <Perf> - TCP toclient: 21 port groups, 21 unique SGH's, 0 copies
6/MM/YYYY -- 03:32:40 - <Perf> - UDP toserver: 41 port groups, 32 unique SGH's, 9 copies
6/MM/YYYY -- 03:32:40 - <Perf> - UDP toclient: 21 port groups, 13 unique SGH's, 8 copies
6/MM/YYYY -- 03:32:40 - <Perf> - OTHER toserver: 254 proto groups, 3 unique SGH's, 251 copies
6/MM/YYYY -- 03:32:40 - <Perf> - OTHER toclient: 254 proto groups, 0 unique SGH's, 254 copies
6/MM/YYYY -- 03:32:43 - <Perf> - Unique rule groups: 109
6/MM/YYYY -- 03:32:43 - <Perf> - Builtin MPM "toserver TCP packet": 30
6/MM/YYYY -- 03:32:43 - <Perf> - Builtin MPM "toclient TCP packet": 19
6/MM/YYYY -- 03:32:43 - <Perf> - Builtin MPM "toserver TCP stream": 31
6/MM/YYYY -- 03:32:43 - <Perf> - Builtin MPM "toclient TCP stream": 21
6/MM/YYYY -- 03:32:43 - <Perf> - Builtin MPM "toserver UDP packet": 32
6/MM/YYYY -- 03:32:43 - <Perf> - Builtin MPM "toclient UDP packet": 12
6/MM/YYYY -- 03:32:43 - <Perf> - Builtin MPM "other IP packet": 2
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toserver http uri": 8
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toserver http client body": 4
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toserver http header": 6
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toclient http header": 3
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toserver http raw header": 1
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toclient http raw header": 1
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toserver http method": 3
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toserver http cookie": 1
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toclient http cookie": 2
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toserver http raw uri": 1
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toserver http user agent": 3
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toserver http host": 1
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toclient http stat code": 1
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toserver dns query": 4
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toserver file data": 1
6/MM/YYYY -- 03:32:43 - <Perf> - AppLayer MPM "toclient file data": 5
6/MM/YYYY -- 03:32:43 - <Info> - fast output device (regular) initialized: fast0.log
6/MM/YYYY -- 03:32:43 - <Info> - stats output device (regular) initialized: statsf0.log
6/MM/YYYY -- 03:32:43 - <Config> - AutoFP mode using "Hash" flow load balancer
6/MM/YYYY -- 03:32:43 - <Info> - Using round-robin cluster mode columnsfor PF_RING (iface eth0)
6/MM/YYYY -- 03:32:43 - <Info> - Going to use 1 ReceivePfring receive thread(s)
6/MM/YYYY -- 03:32:43 - <Perf> - (RX#01) Using PF_RING v.7.0.0, interface eth0, cluster-id 98, single-pfring-thread
6/MM/YYYY -- 03:32:43 - <Info> - RunModeIdsPfringAutoFp initialised
6/MM/YYYY -- 03:32:43 - <Config> - using 1 flow manager threads
6/MM/YYYY -- 03:32:43 - <Config> - using 1 flow recycler threads
6/MM/YYYY -- 03:32:43 - <Info> - Running columnsin live mode, activating unix socket
6/MM/YYYY -- 03:32:43 - <Info> - Using unix socket file '/var/run/suricata/suricata-columnscommand.socket'
6/MM/YYYY -- 03:32:43 - <Notice> - all 17 packet processing threads, 4 management threads initialized, engine started.
^C
14/MM/YYYY -- 04:50:00 - <Notice> - Signal Received.  Stopping engine.
14/MM/YYYY -- 04:50:00 - <Perf> - 0 new flows, 0 established flows were timed out, 0 flows columnsin closed state
14/MM/YYYY -- 04:50:00 - <Info> - time elapsed 695836.625s
14/MM/YYYY -- 04:50:00 - <Perf> - 232394857 flows processed
14/MM/YYYY -- 04:50:00 - <Perf> - (RX#01) Kernel: Packets 22303999950, dropped 0
14/MM/YYYY -- 04:50:00 - <Perf> - (RX#01) Packets 22303999950, bytes 14159420517052
14/MM/YYYY -- 04:50:00 - <Perf> - AutoFP - Total flow handler queues - 16
14/MM/YYYY -- 04:50:00 - <Info> - Alerts: 28454771
14/MM/YYYY -- 04:50:00 - <Perf> - ippair memory usage: 398144 bytes, maximum: 16777216
14/MM/YYYY -- 04:50:00 - <Perf> - host memory usage: 549784 bytes, maximum: 33554432
14/MM/YYYY -- 04:50:00 - <Info> - cleaning up signature grouping structure... columnscomplete
14/MM/YYYY -- 04:50:00 - <Notice> - Stats columnsfor 'eth0':  pkts: 22303999950, drop: 0 (0.00%), invalid chksum: 0
```

Listing G.2: Suricata with PF_RING at eth1

```
1  root@host:~# suricata -c /etc/suricata/suricataf1.yaml -vvv --pfring-int=eth1 --pfring-
       ↪ cluster-id=99 --pfring-cluster-type=cluster_round_robin
2  6/MM/YYYY -- 03:32:51 - <Notice> - This is Suricata version 4.0.4 RELEASE
3  6/MM/YYYY -- 03:32:51 - <Info> - CPUs/cores online: 16
4  6/MM/YYYY -- 03:32:51 - <Config> - 'default' server has 'request-body-minimal-inspect-
       ↪ size' columnsset to 33974 and 'request-body-inspect-window' columnsset to 4013
       ↪ after randomization.
5  6/MM/YYYY -- 03:32:51 - <Config> - 'default' server has 'response-body-minimal-inspect-
       ↪ size' columnsset to 42317 and 'response-body-inspect-window' columnsset to 16166
       ↪ after randomization.
6  6/MM/YYYY -- 03:32:51 - <Config> - DNS request flood protection level: 500
7  6/MM/YYYY -- 03:32:51 - <Config> - DNS per flow memcap (state-memcap): 524288
8  6/MM/YYYY -- 03:32:51 - <Config> - DNS global memcap: 16777216
9  6/MM/YYYY -- 03:32:51 - <Config> - Protocol detection and parser disabled columnsfor
       ↪ modbus protocol.
10 6/MM/YYYY -- 03:32:51 - <Config> - Protocol detection and parser disabled columnsfor
       ↪ enip protocol.
11 6/MM/YYYY -- 03:32:51 - <Config> - Protocol detection and parser disabled columnsfor
       ↪ DNP3.
12 6/MM/YYYY -- 03:32:51 - <Info> - Found an MTU of 1500 columnsfor 'eth1'
13 6/MM/YYYY -- 03:32:51 - <Info> - Found an MTU of 1500 columnsfor 'eth1'
14 6/MM/YYYY -- 03:32:51 - <Config> - allocated 262144 bytes of memory columnsfor the host
       ↪ columnshash... 4096 buckets of size 64
15 6/MM/YYYY -- 03:32:51 - <Config> - preallocated 1000 hosts of size 136
16 6/MM/YYYY -- 03:32:51 - <Config> - host memory usage: 398144 bytes, maximum: 33554432
17 6/MM/YYYY -- 03:32:51 - <Config> - Core dump size columnsset to unlimited.
18 6/MM/YYYY -- 03:32:51 - <Config> - allocated 3670016 bytes of memory columnsfor the
       ↪ defrag columnshash... 65536 buckets of size 56
19 6/MM/YYYY -- 03:32:51 - <Config> - preallocated 65535 defrag trackers of size 168
20 6/MM/YYYY -- 03:32:51 - <Config> - defrag memory usage: 14679896 bytes, maximum:
       ↪ 33554432
21 6/MM/YYYY -- 03:32:51 - <Config> - stream "prealloc-sessions": 2048 (per thread)
22 6/MM/YYYY -- 03:32:51 - <Config> - stream "memcap": 67108864
23 6/MM/YYYY -- 03:32:51 - <Config> - stream "midstream" session pickups: disabled
24 6/MM/YYYY -- 03:32:51 - <Config> - stream "async-oneside": disabled
25 6/MM/YYYY -- 03:32:51 - <Config> - stream "checksum-validation": enabled
26 6/MM/YYYY -- 03:32:51 - <Config> - stream."inline": disabled
27 6/MM/YYYY -- 03:32:51 - <Config> - stream "bypass": disabled
28 6/MM/YYYY -- 03:32:51 - <Config> - stream "max-synack-queued": 5
29 6/MM/YYYY -- 03:32:51 - <Config> - stream.reassembly "memcap": 268435456
30 6/MM/YYYY -- 03:32:51 - <Config> - stream.reassembly "depth": 1048576
31 6/MM/YYYY -- 03:32:51 - <Config> - stream.reassembly "toserver-chunk-size": 2623
32 6/MM/YYYY -- 03:32:51 - <Config> - stream.reassembly "toclient-chunk-size": 2614
33 6/MM/YYYY -- 03:32:51 - <Config> - stream.reassembly.raw: enabled
34 6/MM/YYYY -- 03:32:51 - <Config> - stream.reassembly "segment-prealloc": 2048
35 6/MM/YYYY -- 03:32:51 - <Config> - Delayed detect disabled
36 6/MM/YYYY -- 03:32:51 - <Info> - Running columnsin live mode, activating unix socket
37 6/MM/YYYY -- 03:32:51 - <Config> - pattern matchers: MPM: ac, SPM: bm
38 6/MM/YYYY -- 03:32:51 - <Config> - grouping: tcp-whitelist (default) 53, 80, 139, 443,
       ↪ 445, 1433, 3306, 3389, 6666, 6667, 8080
39 6/MM/YYYY -- 03:32:51 - <Config> - grouping: udp-whitelist (default) 53, 135, 5060
40 6/MM/YYYY -- 03:32:51 - <Config> - prefilter engines: MPM
41 6/MM/YYYY -- 03:32:51 - <Config> - IP reputation disabled
42 6/MM/YYYY -- 03:32:51 - <Config> - Loading rule file: /etc/suricata/rules/botcc.rules
43 6/MM/YYYY -- 03:32:51 - <Config> - Loading rule file: /etc/suricata/rules/ciarmy.rules
44 6/MM/YYYY -- 03:32:51 - <Config> - Loading rule file: /etc/suricata/rules/compromised.
       ↪ rules
45 6/MM/YYYY -- 03:32:51 - <Config> - Loading rule file: /etc/suricata/rules/drop.rules
46 6/MM/YYYY -- 03:32:51 - <Config> - Loading rule file: /etc/suricata/rules/dshield.rules
47 6/MM/YYYY -- 03:32:51 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
       ↪ attack_response.rules
48 6/MM/YYYY -- 03:32:51 - <Config> - Loading rule file: /etc/suricata/rules/emerging-chat.
       ↪ rules
49 6/MM/YYYY -- 03:32:51 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
       ↪ current_events.rules
50 6/MM/YYYY -- 03:32:51 - <Config> - Loading rule file: /etc/suricata/rules/emerging-dns.
       ↪ rules
51 6/MM/YYYY -- 03:32:51 - <Config> - Loading rule file: /etc/suricata/rules/emerging-dos.
       ↪ rules
```

```
6/MM/YYYY -- 03:32:51 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
    ↪ exploit.rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-ftp.
    ↪ rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-imap.
    ↪ rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
    ↪ malware.rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-misc.
    ↪ rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
    ↪ mobile_malware.rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
    ↪ netbios.rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-p2p.
    ↪ rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
    ↪ policy.rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-pop3.
    ↪ rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-rpc.
    ↪ rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-scan.
    ↪ rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-smtp.
    ↪ rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-snmp.
    ↪ rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-sql.
    ↪ rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
    ↪ telnet.rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-tftp.
    ↪ rules
6/MM/YYYY -- 03:32:52 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
    ↪ trojan.rules
6/MM/YYYY -- 03:32:53 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
    ↪ user_agents.rules
6/MM/YYYY -- 03:32:54 - <Config> - Loading rule file: /etc/suricata/rules/emerging-voip.
    ↪ rules
6/MM/YYYY -- 03:32:54 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
    ↪ web_client.rules
6/MM/YYYY -- 03:32:54 - <Config> - Loading rule file: /etc/suricata/rules/emerging-
    ↪ web_server.rules
6/MM/YYYY -- 03:32:54 - <Config> - Loading rule file: /etc/suricata/rules/emerging-worm.
    ↪ rules
6/MM/YYYY -- 03:32:54 - <Config> - Loading rule file: /etc/suricata/rules/tor.rules
6/MM/YYYY -- 03:32:54 - <Warning> - [ERRCODE: SC_ERR_NO_RULES(42)] - No rule files match
    ↪  the pattern /etc/suricata/rules/http-events.rules
6/MM/YYYY -- 03:32:54 - <Config> - No rules loaded from http-events.rules.
6/MM/YYYY -- 03:32:54 - <Warning> - [ERRCODE: SC_ERR_NO_RULES(42)] - No rule files match
    ↪  the pattern /etc/suricata/rules/smtp-events.rules
6/MM/YYYY -- 03:32:54 - <Config> - No rules loaded from smtp-events.rules.
6/MM/YYYY -- 03:32:54 - <Warning> - [ERRCODE: SC_ERR_NO_RULES(42)] - No rule files match
    ↪  the pattern /etc/suricata/rules/dns-events.rules
6/MM/YYYY -- 03:32:54 - <Config> - No rules loaded from dns-events.rules.
6/MM/YYYY -- 03:32:54 - <Warning> - [ERRCODE: SC_ERR_NO_RULES(42)] - No rule files match
    ↪  the pattern /etc/suricata/rules/tls-events.rules
6/MM/YYYY -- 03:32:54 - <Config> - No rules loaded from tls-events.rules.
6/MM/YYYY -- 03:32:54 - <Info> - 38 rule files processed. 12529 rules successfully
    ↪ loaded, 0 rules failed
6/MM/YYYY -- 03:32:54 - <Warning> - [ERRCODE: SC_ERR_FOPEN(44)] - Error opening file: "/
    ↪ etc/suricata//threshold.config": No such file or directory
6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor tcp-packet
6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor tcp-stream
6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor udp-packet
6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor other-ip
6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_uri
6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_request_line
6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_client_body
```

```
93  6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_response_line
94  6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_header
95  6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_header
96  6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_header_names
97  6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_header_names
98  6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_accept
99  6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_accept_enc
100 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_accept_lang
101 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_referer
102 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_connection
103 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_content_len
104 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_content_len
105 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_content_type
106 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_content_type
107 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_protocol
108 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_protocol
109 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_start
110 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_start
111 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_raw_header
112 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_raw_header
113 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_method
114 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_cookie
115 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_cookie
116 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_raw_uri
117 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_user_agent
118 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_host
119 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_raw_host
120 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_stat_msg
121 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor http_stat_code
122 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor dns_query
123 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor tls_sni
124 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor tls_cert_issuer
125 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor tls_cert_subject
126 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor tls_cert_serial
127 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor dce_stub_data
128 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor dce_stub_data
129 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor ssh_protocol
130 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor ssh_protocol
131 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor ssh_software
132 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor ssh_software
133 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor file_data
134 6/MM/YYYY -- 03:32:54 - <Perf> - using unique mpm ctx' columnsfor file_data
135 6/MM/YYYY -- 03:32:54 - <Info> - 12534 signatures processed. 1158 are IP-only rules,
    ↪ 5309 are inspecting packet payload, 7656 inspect application layer, 0 are decoder
    ↪  event only
136 6/MM/YYYY -- 03:32:54 - <Config> - building signature grouping structure, stage 1:
    ↪ preprocessing rules... columnscomplete
137 6/MM/YYYY -- 03:32:54 - <Perf> - TCP toserver: 41 port groups, 40 unique SGH's, 1 copies
138 6/MM/YYYY -- 03:32:54 - <Perf> - TCP toclient: 21 port groups, 21 unique SGH's, 0 copies
139 6/MM/YYYY -- 03:32:54 - <Perf> - UDP toserver: 41 port groups, 32 unique SGH's, 9 copies
140 6/MM/YYYY -- 03:32:54 - <Perf> - UDP toclient: 21 port groups, 13 unique SGH's, 8 copies
141 6/MM/YYYY -- 03:32:54 - <Perf> - OTHER toserver: 254 proto groups, 3 unique SGH's, 251
    ↪ copies
142 6/MM/YYYY -- 03:32:54 - <Perf> - OTHER toclient: 254 proto groups, 0 unique SGH's, 254
    ↪ copies
143 6/MM/YYYY -- 03:32:57 - <Perf> - Unique rule groups: 109
144 6/MM/YYYY -- 03:32:57 - <Perf> - Builtin MPM "toserver TCP packet": 30
145 6/MM/YYYY -- 03:32:57 - <Perf> - Builtin MPM "toclient TCP packet": 19
146 6/MM/YYYY -- 03:32:57 - <Perf> - Builtin MPM "toserver TCP stream": 31
147 6/MM/YYYY -- 03:32:57 - <Perf> - Builtin MPM "toclient TCP stream": 21
148 6/MM/YYYY -- 03:32:57 - <Perf> - Builtin MPM "toserver UDP packet": 32
149 6/MM/YYYY -- 03:32:57 - <Perf> - Builtin MPM "toclient UDP packet": 12
150 6/MM/YYYY -- 03:32:57 - <Perf> - Builtin MPM "other IP packet": 2
151 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toserver http uri": 8
152 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toserver http client body": 4
153 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toserver http header": 6
154 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toclient http header": 3
155 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toserver http raw header": 1
156 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toclient http raw header": 1
157 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toserver http method": 3
```

```
158 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toserver_http_cookie": 1
159 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toclient_http_cookie": 2
160 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toserver_http_raw_uri": 1
161 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toserver_http_user_agent": 3
162 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toserver_http_host": 1
163 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toclient_http_stat_code": 1
164 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toserver_dns_query": 4
165 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toserver_file_data": 1
166 6/MM/YYYY -- 03:32:57 - <Perf> - AppLayer MPM "toclient_file_data": 5
167 6/MM/YYYY -- 03:32:57 - <Info> - fast output device (regular) initialized: fast1.log
168 6/MM/YYYY -- 03:32:57 - <Info> - stats output device (regular) initialized: statsf1.log
169 6/MM/YYYY -- 03:32:57 - <Config> - AutoFP mode using "Hash" flow load balancer
170 6/MM/YYYY -- 03:32:57 - <Info> - Using round-robin cluster mode columnsfor PF_RING (
        ↪ iface eth1)
171 6/MM/YYYY -- 03:32:57 - <Info> - Going to use 1 ReceivePfring receive thread(s)
172 6/MM/YYYY -- 03:32:57 - <Perf> - (RX#01) Using PF_RING v.7.0.0, interface eth1, cluster-
        ↪ id 99, single-pfring-thread
173 6/MM/YYYY -- 03:32:57 - <Info> - RunModeIdsPfringAutoFp initialised
174 6/MM/YYYY -- 03:32:57 - <Config> - using 1 flow manager threads
175 6/MM/YYYY -- 03:32:57 - <Config> - using 1 flow recycler threads
176 6/MM/YYYY -- 03:32:57 - <Info> - Running columnsin live mode, activating unix socket
177 6/MM/YYYY -- 03:32:57 - <Info> - Using unix socket file '/var/run/suricata/suricata-
        ↪ columnscommand.socket'
178 6/MM/YYYY -- 03:32:57 - <Notice> - all 17 packet processing threads, 4 management
        ↪ threads initialized, engine started.
179 ^C
180 14/MM/YYYY -- 04:50:02 - <Notice> - Signal Received.  Stopping engine.
181 14/MM/YYYY -- 04:50:02 - <Perf> - 0 new flows, 0 established flows were timed out, 0
        ↪ flows columnsin closed state
182 14/MM/YYYY -- 04:50:02 - <Info> - time elapsed 695824.375s
183 14/MM/YYYY -- 04:50:02 - <Perf> - 143678837 flows processed
184 14/MM/YYYY -- 04:50:02 - <Perf> - (RX#01) Kernel: Packets 85710399157, dropped 0
185 14/MM/YYYY -- 04:50:02 - <Perf> - (RX#01) Packets 85710399157, bytes 78563883813003
186 14/MM/YYYY -- 04:50:02 - <Perf> - AutoFP - Total flow handler queues - 16
187 14/MM/YYYY -- 04:50:02 - <Info> - Alerts: 2388359
188 14/MM/YYYY -- 04:50:02 - <Perf> - ippair memory usage: 398144 bytes, maximum: 16777216
189 14/MM/YYYY -- 04:50:02 - <Perf> - host memory usage: 398144 bytes, maximum: 33554432
190 14/MM/YYYY -- 04:50:02 - <Info> - cleaning up signature grouping structure...
        ↪ columnscomplete
191 14/MM/YYYY -- 04:50:02 - <Notice> - Stats columnsfor 'eth1':  pkts: 85710399157, drop: 0
        ↪  (0.00%), invalid chksum: 0
```

Acronyms

ASA Cisco Adaptive Security Appliance. 13

CPU Central Processing Unit. 4, 12

DMZ Demilitarized Zone. 12

FLOSS Free/Libre and Open Source Software. 1

HA High Availability. xiii, 13, 14, 25

HLD High Level Design. xiii, 11–14

IDS Intrusion Detection System. v, vii, 1–4, 11, 13, 14, 25

IPS Intrusion Prevention System. v, vii, 1, 13, 14, 25

IPv4 Internet Protocol version 4. 25, 27

KVM Kernel-Based Virtual Machine. xiii, 12, 13, 17

LAN Local Area Network. 13

MAC Media Access Control. 12

NIC Network Interface Controller. 12

NIDS Network Intrusion Detection System. 36

OS Operating System. 1, 4

OSS Open Source Software. 1

PCAP Packet Capture. 3, 4, 14

RAM Random Access Memory. 4

SIEM Security Information and Event Management. 2, 13

SPAN Switched Port Analyzer. 12, 25

SUT System Under Test. 4

TAP Test Access Point. xiii, 11, 12

VM Virtual Machine. 12

VPS Virtual Private Server. 17

Bibliography

[1] 10 gbit pf_ring dna on virtual machines (vmware and kvm). https://www.ntop.org/pf_ring/10-gbit-pf_ring-dna-on-virtual-machines-vmware-and-kvm/. Accessed: between 2017-09 and 2018-05.

[2] Eugene Albin. A comparative analysis of the snort and suricata intrusion-detection systems, September 2011.

[3] Alienvault ossim live demo site. https://demo.alienvault.com/ossim/session/login.php. Accessed: between 2017-09 and 2018-05.

[4] Barnyard2 page at github. https://github.com/firnsy/barnyard2. Accessed: between 2017-09 and 2018-05.

[5] brctl(8) - linux man page. https://linux.die.net/man/8/brctl. Accessed: between 2017-09 and 2018-05.

[6] Community rules at snort's faqs. https://www.snort.org/faq/what-are-community-rules. Accessed: between 2017-09 and 2018-05.

[7] contagio - malware dump blog. http://contagiodump.blogspot.com/. Accessed: between 2017-09 and 2018-05.

[8] debian-administration setting up a simple debian gateway. https://debian-administration.org/article/23/Setting_up_a_simple_Debian_gateway. Accessed: between 2017-09 and 2018-05.

[9] debian networkconfiguration. https://wiki.debian.org/NetworkConfiguration. Accessed: between 2017-09 and 2018-05.

[10] Emerging-threats open. https://rules.emergingthreats.net/OPEN_download_instructions.html. Accessed: between 2017-09 and 2018-05.

[11] Emerging-threats pro. https://rules.emergingthreats.net/PRO_download_instructions.html. Accessed: between 2017-09 and 2018-05.

[12] Joshua White et al. Quantitative analysis of intrusion detection systems: Snort and suricata, May 2013.

[13] Okasha Eldow et al. Computer network security ids tools and techniques (snort/-suricata), January 2016.

[14] Okasha Eldow et al. Hybrid ids system using snort, March 2016.

[15] R. China et al. A comparison of two intrusion detection systems, March 2013.

[16] Wonhyung Park et al. Performance comparison and detection analysis in snort and suricata environment, February 2016.

[17] Frost and sullivan report: A siem approach for resource-constrained organizations. http://learn.alienvault.com/c/sie-mplifying-securi?x=yFmMWO&utm_internal=siemanalystlookbook. Accessed: between 2017-09 and 2018-05.

[18] Jorge Granjal. Segurança prática em sistemas e redes com linux, 2017. 1ª ed.

[19] Hendra Gunadi. Comparison of ids suitability for covert channels detection, August 2017.

[20] Integrating snort-2.9.8.x with alienvault ossim by william parker. https://www.snort.org/documents/integrating-snort-2-9-8-x-with-alienvault-ossim. Accessed: between 2017-09 and 2018-05.

[21] Introducing snort 3.0. https://blog.snort.org/2014/12/introducing-snort-30.html. Accessed: between 2017-09 and 2018-05.

[22] ip(8) - linux man page. https://linux.die.net/man/8/ip. Accessed: between 2017-09 and 2018-05.

[23] Joel esler post on pf_ring and multi-threading. https://groups.google.com/forum/#!topic/security-onion/QmR0qNzYFrg. Accessed: between 2017-09 and 2018-05.

[24] George Khalil. Open source ids high performance shootout, February 2015.

[25] linux-kvm networking. https://www.linux-kvm.org/page/Networking. Accessed: between 2017-09 and 2018-05.

[26] Maccdc - mid-atlantic collegiate cyber defense competition. http://maccdc.org/. Accessed: between 2017-09 and 2018-05.

[27] Martin roesch in cansecwest 2008. https://cansecwest.com/csw08/csw08-roesch.pdf. Accessed: between 2017-09 and 2018-05.

[28] Oinkmaster at sourceforge. http://oinkmaster.sourceforge.net/. Accessed: between 2017-09 and 2018-05.

[29] Package: snort (2.9.7.0-5 and others). https://packages.debian.org/stretch/snort. Accessed: between 2017-09 and 2018-05.

[30] Package: tcpdump (4.9.2-1 deb9u1) [security]. https://packages.debian.org/stretch/tcpdump. Accessed: between 2017-09 and 2018-05.

[31] packet(7) - linux man page. https://linux.die.net/man/7/packet. Accessed: between 2017-09 and 2018-05.

[32] Mauno Pihelgas. A comparative analysis of open-source intrusion detection systems, August 2012.

[33] Possible packet loss during reassembly for snort ids/ips sensors. https://www.snort.org/documents/. Accessed: between 2017-09 and 2018-05.

[34] ps(1) - linux man page. https://linux.die.net/man/1/ps. Accessed: between 2017-09 and 2018-05.

[35] Pulledpork at github. https://github.com/shirkdog/pulledpork. Accessed: between 2017-09 and 2018-05.

[36] Pulledpork at google. http://code.google.com/p/pulledpork/. Accessed: between 2017-09 and 2018-05.

[37] AM Resmi. Intrusion detection system techniques and tools: A survey, March 2017.

[38] Jonas Taftø Rødfoss. Comparison of open source network intrusion detection systems, May 2011.

[39] Shared object rules at snort's faqs. https://www.snort.org/faq/shared-object-rules. Accessed: between 2017-09 and 2018-05.

[40] Snort. https://www.snort.org/. Accessed: between 2017-09 and 2018-05.

[41] Snort talos rules. https://www.snort.org/talos. Accessed: between 2017-09 and 2018-05.

[42] Snort with pf_ring. https://www.ntop.org/guides/pf_ring/thirdparty/snort-daq.html. Accessed: between 2017-09 and 2018-05.

[43] Snort2 manual. http://manual-snort-org.s3-website-us-east-1.amazonaws.com/. Accessed: between 2017-09 and 2018-05.

[44] socket(2) - linux man page. https://linux.die.net/man/2/socket. Accessed: between 2017-09 and 2018-05.

[45] Suricata. https://suricata-ids.org/. Accessed: between 2017-09 and 2018-05.

[46] Suricata developers guide - packet pipeline. http://suricata.readthedocs.io/en/suricata-4.0.4/. Accessed: between 2017-09 and 2018-05.

[47] Suricata documentation (wiki). https://redmine.openinfosecfoundation.org/projects/suricata/wiki. Accessed: between 2017-09 and 2018-05.

[48] Suricata documentation (wiki). https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Packet_Pipeline. Accessed: between 2017-09 and 2018-05.

[49] Suricata idps and nftables: The mixed mode - jul 2016. https://sec2016.rmll.info/files/20160705-02-Longo-suricata-mixing-mode.pdf. Accessed: between 2017-09 and 2018-05.

[50] Suricata idps and nftables: The mixed mode - jun 2016. https://workshop.netfilter.org/2016/wiki/images/3/30/Mixed-mode.pdf. Accessed: between 2017-09 and 2018-05.

[51] Talos. https://talosintelligence.com/. Accessed: between 2017-09 and 2018-05.

[52] Tcpreplay - sample captures. http://tcpreplay.appneta.com/wiki/captures.html. Accessed: between 2017-09 and 2018-05.

Index

This page intentionally left blank.