Quantitative Analysis Of Intrusion Detection Systems: Snort and Suricata

Joshua White, Thomas Fitzsimmons, James Licata, and Jeanna Matthews

Clarkson University, Potsdam NY 13676, USA,
whitejs, fitzsitd, licataja, jnm@clarkson.edu

Abstract. Any modern organization that is serious about security deploys a network intrusion detection system (NIDS) to monitor network traffic for signs of malicious activity. The most widely deployed NIDS system is Snort, an open source system originally released in 1998. Snort is a single threaded system that uses a set of clear text rules to instruct a base engine how to react when particular traffic patterns are detected. In 2009, the US Department of Homeland Security and a consortium of private companies provided substantial grant funding to a newly created organization known as the Open Information Security Foundation (OISF), to build a multi-threaded alternative to Snort, called Suricata. Despite many similarities between Snort and Suricata, the OISF stated it was essential to replace the older single-threaded Snort engine with a multi-threaded system that could deliver higher performance and better scalability. Key Snort developers argued that Suricata's multi-threaded architecture would actually slow the detection process. Given these competing claims, an objective head-to-head comparison of the performance of Snort and Suricata is needed. In this paper, we present a comprehensive quantitative comparison of the two systems. We have developed a rigorous testing framework that examines the performance of both systems as we scale system resources. Our results show that a single instance of Suricata is able to deliver substantially higher performance than a corresponding single instance of Snort, but has problems scaling with a higher number of cores. We find that while Suricata needs tuning for a higher number of cores, it is still able to surpass Snort even at 1 core where we would have expected Snort to shine.

Keywords: Network Intrusion Detection Systems, Snort, Suricata, OISF, Security, Multi-threaded Performance, Scaling

1 Introduction

Network Intrusion Detection Systems (NIDS) capture and inspect network traffic for signs of malicious or unauthorized activity. NIDS can be compared on many dimensions including performance, scalability, ability to detect different kinds of attacks, ease of use, the responsiveness of the development community to requests for new features, documentation quality, and many others. In this paper, we focus specifically on comparing the performance and scalability of two

of the most widely deployed NIDS: Snort and Suricata. Keeping up with all the traffic on a busy network is a performance intensive activity. If the NIDS is unable to keep up with the traffic in real-time, then uninspected packets are either dropped causing problems for legitimate traffic or allowed to flow causing problems for security.

In both Snort and Suricata, a base engine is controlled by a set of rules. Each rule describes network activity that is considered malicious or unwanted by specifying the content of network packets. Each rule also specifies an action to be taken in case a packet is suspect, such as raising an alert or dropping the packet. The base engine must read each raw packet from the network and quickly compare it against all the rules loaded into the system to determine what action to take.

In this section, we expand a bit on the history of both Snort and Suricata.

1.1 Snort

Snort is a free open source, NIDS. Originally released in 1998 by Martin Roesch as a lightweight cross-platform network sniffing tool (around 1200 lines of code), it has evolved into a powerful and full-featured intrusion detection and prevention product. Snort is a successful example of the open source development methodology in which community members contribute source code, bug reports, bug fixes, documentation and related tools.

In November 1999, Roesch published "Snort: Lightweight Intrusion Detection for Networks" at the 13th Annual LISA Conference. He detailed his work creating a pattern matching system for the output of the Snort sniffer [2]. Soon after preprocessors for protocol normalization where added to the Snort engine. This allowed a single rule to be applied to any variation of a protocol.

Today, Snort is one the most popular security tools of all time [6, 7, 4, 5]. According to the Snort web site, it is actually the most widely deployed intrusion prevention technology in the world [3].

Sourcefire Inc., a company founded in 2001 by Martin Roesch, manages the development of Snort and offers commercial products based on Snort. They also research the newest threats and develop rules for detecting them. The newest rules are available by paid subscription immediately and then released to all registered users after 30 days.

1.2 Suricata

In 2009, the US Department of Homeland Security, along with a consortium of private companies, provided substantial grant funding to a newly created organization, the Open Information Security Foundation (OISF). The grant was to build an alternative to Snort, called Suricata. Suricata was first released in 2010 and the current version 1.1.1 was released in December 2011.

Although all code is original, Suricata developers have made no attempt to disguise the many ways in which they are borrowing from the Snort architecture. They readily acknowledge Snort as "our collective roots". Suricata can even be used with the same rule sets used by Snort.

1.3 Snort vs. Suricata

With the wide success of Snort, it is natural to wonder what would motivate the development of another similar open source system. One of the primary reasons was concern for the performance limits of Snort's single threaded architecture. When Snort was built, it was designed to run on the most popular computers of the time, 32-bit single core systems. While many improvements have been made to Snort over the past 14 years, the base engine has remained single-threaded.

The task of comparing multiple network packets against a large list of intrusion detection rules certainly appears to be a highly parallelizable task for which modern multi-core systems would be well suited. Still, there has been substantial public debate about the need for a multi-threaded engine. OISF president, Matt Jonkman, argues that a multi-threaded engine is essential for high performance and scalability, but that there has been no commercial motivation for anyone in the Snort community to invest in rewriting the core engine from scratch. [22] He stated that "the money goes to management/forensics consoles, rules, and big fast boxes" [23]. On the other hand, Martin Roesch, founder of Snort and Sourcefire, argued that the multi-threaded architecture would actually slow detection rather than make it faster [22]. The performance of Suricata's specific implementation of a multi-threaded architecture has also been criticized. For example, Roesch had called Suricata "a clone of Snort that performs worse at taxpayer's expense" and SourceFire's Vulnerability Research Team has reported that the performance of Suricata "isn't just bad; it's hideously, unforgivably bad" [23].

Beyond questions about performance and the fundamental need for a multi-threaded architecture, Matt Jonkman, OISF founder, cites concerns about the Snort development process [25] as significant factors in the development of Suricata. [1] Unlike many open source projects that have commercial offerings, Sourcefire has not forked a separate commercial version of Snort under a dual-license model [26]. As a result the community version of Snort is tightly controlled by Sourcefire and it is difficult to get insight into the Snort development process, bug tracking information or future roadmap. In addition, developers outside Sourcefire have a difficult time submitting bug fixes. Jonkman offers up the example of Will Metcalf, now OISF QA lead, who discovered 35+ major bugs in

[1] Jonkman clarifies that the various reasons cited for the development of Suricata are not necessarily the same as what motivated the Department of Homeland Security (DHS) and many private companies to contribute backing to the project[25]. Instead, it was both government and industry need for innovation in the IDS world [27].

Snort over the past 3 years. When Metcalf tried to contribute code to fix these bugs, he reports that his contributions went unacknowledged by the Snort development team and that a number of the flaws still remain. Any developers that do succeed in contributing code to Snort must also sign over the rights to that code to Sourcefire rather than simply releasing it under an open source license. In the case of Suricata, the OISF owns the copyright and the software can not be commercialized because the foundation is a non-profit organization [25]. [2]

While the focus of this paper is performance and scalability, it is also worth briefly mentioning some other differences between Snort and Suricata. There are certainly differences in features. For example, Suricata includes some innovative features such as the ability to automatically detect common network protocols even when they are used on non-standard ports. Similarly, Snort contains features unavailable in Suricata, including some features required in classified environments such as the option of hiding the rules being used for inspecting network traffic. In addition, since Suricata is newer and less widespread, there are fewer resources documenting installation procedures and configuration options than there are for Snort.

2 Related Work

Given the contentious debate regarding performance of Snort and Suricata, it is surprising that little side-by-side comparison data has been published. One paper with similar objectives to our own was "A Comparitive Analysis of Snort and Suricata Intrusion Detection Systems" by Eugene Albin [13]. Albin presents three experiments in comparing the performance of Snort and Suricata: using live network traffic, static pcap files, and testing ruleset functionality using Pytbull. Albin used a VMware ESXi hosted virtual machine for the majority of his work and states that this may have contributed to the potential skewing of packet rates when compared to a physical machine. The systems were tested in a network environment where Snort was already deployed and tuned for the hardware resources. This may have given Snort an advantage.

One of the chief differences between Albin's work and this research, was an unquantified and varying component of background traffic that limits the repeatability of his tests. For example, Albin's second experiment shows a graph comparing the performance of both Snort and Suricata showing live network traffic varied between runs. Albin points out that there was no consistency across runs in terms of traffic type and traffic volume. Albin's use of PytBull in his later experiments also appears to have also been done while connected to a live network, causing traffic encountered to be potentially susceptible to the same inherent network variability.

[2] The OISF views Suricata as the first of many projects and intends to be a safe place for long term open source projects to reside without developers needing to fear that the project will become closed or their contributions unacknowledged.

3 Methodology

The goal of this paper is to present a thorough, repeatable, quantitative, apples-to-apples comparison of the performance of Snort and Suricata. Our experiments were conducted on a flexible hardware platform that allowed us to scale the hardware resources actively available. We ran our experiments with pcap traces to ensure repeatability and are providing all of our scripts and traces to encourage others to run the test suite on their own hardware.

In this section, we describe in detail the ways in which we varied each aspect of our testing. We varied the number of cores used, the rulesets used, the configuration of each IDS and the workload used. The script we developed for our testing consists of approximately 3000 LOC and is available at our project webpage [21]. Our basic methodology could easily be extended to include more workloads via additional pcap traces, additional rulesets and additional configuration settings. The same test suite could also be used on any hardware platform for which Snort and Suricata are available.

For each test, we capture a variety of metrics including packets per second (PPS) as processed by each IDS, the amount of memory used by the IDS process(es) and the CPU utilization. To capture PPS measurements, we use builtin reporting functions in each IDS. For Snort, we enabled the perfmonitor function as described in the perfmon performance profiling guide [35]. Suricata has a similar functionality known as statistical logging that is enabled by default. Using the snort.conf and suricata.yaml files we set both engines to output statistics to their respective log files in 1 second intervals. To capture memory usage and CPU utilization measurements, our scripts parsed the output of the standard ps command on Linux and extracted the memory usage and CPU usage information for the IDS process(es).

3.1 Varying the Hardware Resources

The test system consisted of an AMD 8439 Opteron class processor and a 4 socket by 4 Memory Bank motherboard. Table 1 describes the details of the hardware platform we used. We were able to vary the number of CPU cores actively used in each experiment.

We would like to encourage others to use our testing scripts and pcap files to run tests in their own hardware environment.This is the best way to gauge actual performance differences before choosing an IDS system.

Table 1. Details Of Hardware Testing Platform

White Box Super Micro Server	
RAM	48 GB DDR2 Memory (ECC)
CPU	24 Core AMD (4 x 6 Core Opteron 8439 2.8 GHz)
CPU Bus	AMD Hypertransport 57.6 GB/ps Comm
Memory Controller	NUMA 12.8 GB/ps to closest 12GB Memory Bank
L1 Cache	128 KB Per Core
L2 Cache	512 KB Per Core
L3 Cache	6 MB Per CPU, shared
Disk	1.5 TB SATA II

3.2 Varying the Rulesets

Emerging Threats is an open source community that was initially created to support an open Snort ruleset. Currently this group produces rulesets compatible with both Snort and Suricata. The Emerging Threats Open Ruleset (ET-Open) consists of contributions from community members and is freely available for download [8].

Emerging Threats, also produces a professional ruleset (ET-Pro). In the ET-Pro ruleset, each item contains a rule portion that is optimized for Snort, a rule portion that is optimized for Suricata and an alert portion that is shared by both engines. Items in the ET-Pro set do not necessarily migrate to the ET-Open set overtime. They are a seperate set that are optimized for Snort and Suricata by the Emerging Threats team. A home user license for the ET-Pro ruleset is currently $35 annually and a enterprise user license is $500 annually per sensor. They offer volume licenses on a case by case basis for organizations requiring over 10 sensors.

The Sourcefire Vulnerability Research Team (VRT) produces the official rule set for Snort. New rules released by the VRT are free to the community after approximately 1 month. They are available immediately upon release through various subscription models. Personal licenses cost $29.99 and business licenses start at $399/sensor [28].

Table 2 summarizes the rulesets we used in our testing.

Table 2. Ruleset Details

Ruleset Name	Version	# Of Rules
ET-Open Ruleset	6953	16179
ET-Pro Ruleset	8101424752816199	13154
Snort VRT Ruleset	Nov 2011	18038

3.3 Varying the Configuration

We tested both Snort and Suricata in their default, out-of-the-box configuration as a baseline. We also experimented with varying some of the configurarion settings. In the results section, we refer to performance optimized configurations of both Snort and Suricata and in thise section we describe in detail what that entails.

In the case of Suricata, configuration changes were done by changing some end-user configuration parameters in the suricata.yaml file and not by compiling in any special acceleration options, such as PF_Ring, that may have added additional performance. The parameter max-pending-packets specifies the maximum number of packets that Suricata can process simultaneously. It has a default value of 50. In the performance optimized tests of Suricata, we set this value to its maximum 65535 packets. This setting is the maximum hard limit for this value due to the packet pool being a lockless ringbuffer that can contain USHRT_MAX. USHRT_MAX is the maximum value that can be stored in an unsigned short variable. Setting this value to its maximum is supposed to increase performance substantially on a multi-core/threaded system as indicated in discussions on the Suricata wiki/mailing lists [13, 15, 18, 17, 19].

Suricata additionally allows for the run-modes to be changed. In the performance optimized tests of Suricata, we have changed the run-mode from its default value of auto to autofp. AutoFP or Automatic Flow Pinned mode is an Intel technology for a multi-threaded environment that can guarantee all processes related to a single packet of data reside on a single core [17]. If a packet requires multiple threads either for pre-processing or rule comparison, overall performance can be hindered if the threads reside on physically different cores. This is caused by time delays necessary for copying the data between cores. AutoFP ensures that if packets are all part of a single flow they will be processed on the same core if the process is specifically doing flow based work.

In the case of Snort, we switched the ac-bnfa-nq (Aho-Corasick Binary NFA) mode that Snort uses to search the payload for specific strings to simple ac (Aho-Corasick) mode. According to many sources this method uses more memory then the default mode but when used can increase performance significantly [29, 30, 31, 32]. We also modified Snorts max-pkt-time and set it to 1000, which mean that any packet taking more then 1000 usec to process is dropped. In many

situations when a network has large bursts of traffic or sensors don't have enough available memory this modification could potentially result in a large number of dropped packets [29, 33, 34].

3.4 Varying the workload

In order to vary the workload presented to the IDS, we used a variety of pcap trace files. The pcap traces we used came from two sources. Our largest data set consisted of the 2010 iCTF Conference "Attacks Against Litya" network capture that consisted of 67GB of captured network traffic (23.7 GB compressed). This network capture was taken during the conference contest in which participants attacked the fictitious "nation of Litya, ruled by the evil Lisboy Bironulesk." [14] The scenario and network service design forced the participants to attack the infrastructure of Litya, much like a critical infrastructure nation state cyber attack.

The second set of pcap files we created outselves by using tcpdump to capture runs of specific PytBull traffic [9]. Pytbull is an open source IDS testing framework. It allows the user to test specific payloads and traffic against different IDS and different rulesets. Pytbull was used to target specified payloads, allowing us to analyze the performance of Snort versus Suricata. The traffic generated by each of the tests were captured in pcap format and replayed for each configuration by our testing script.

Table 3 summarizes the set of pcap files we used in our testing.

Table 3. PCap File Details

pcap Test Name	**Description**	**# Of Packets**	**Size**
iCTF Conf. Workload	Real Production Workload	818178784	67 GB
Client-side Attacks	Client-side Download Attacks	3786	6.1 MB
Test Rules	Basic Ruleset Testing	245167	22.5 MB
Bad Traffic	Non-RFC Compliant Packets	2152	3.7 MB
Fragmented Packets	Various Fragmented Payloads	2459	4.4 MB
Multiple Failed Logins	Track Multiple Failed Logins	1200	1.8 MB
Evasion Techniques (ET)	Detection of Various (ET)	52323	21.6 MB
Shell Codes	Ability to Detect Shell Codes	8200	15.4 MB
Denial Of Service	Ability to Detect DoS Attempts	3312	2.3 MB
Pytbull All	TCPDump of All Pytbull Runs	308067	63.2 MB

4 Results

We ran a total of 8600 tests. Specifically, we tested 10 workloads (as shown in Table 3), 4 rulesets (ET-Open, ET-Pro, VRT and No ruleset), 4 configurations

(default and performance optimzied configurations of both Snort and Suricata as discussed in section 3.3) and 10 settings for the number of available cores (1,2,3,4,5,6,8,12,18,and 24). Each of these 1600 variations were run 5 times each. In addition to these 8000 tests in which each IDS processed packets directly from the pcacp files, we also ran another 600 tests in which we used a separate machine to read the pcap files and play them onto the network segment for processing by the IDS.

At the time of our testing, we used the most recent versions of Snort and Suricata were pulled from available repositories and used. The versions used were as follows: Snort v2.8.5.2, and Suricata v1.0.2.

4.1 Snort vs. Suricata Defaults

Figure 1 shows a baseline comparison of Snort and Suricata. Both Snort and Suricata are shown in their default out-of-the-box configuration and they are both using the exact same set of rules, the ET-Open Ruleset. The workload used is the 2010 iCTF Conference trace.

The y-axis shows the average Packets Per Second (PPS)while the x-axis shows the number of CPU cores used. Based on Albin's study and on other known opinions about the performance of Snort and Suricata, we expected that Suricata would not perform as well as the mature Snort on a single core, but would eventually out-perform Snort as additional cores were made available. However, Suricata outperformed Snort even at 1 core. Initial releases of Suricata may have had lower performance, but if so, these results indicate substantial efforts at tuning.

Snort's performance is flat as we add additional cores because the single threaded architecture is unable to take advantage of the additional cores. Suricata's performance initially increases as cores are added, but peaks between 2 and 4 cores and then drops off substantially as more cores are added. This does not reflect well on Suricata's promise of high scalability. In some sense, it is not surprising that Suricata is tuned for 2-4 cores as that is a common configuration. However, we would not have expected to see performance decrease as more cores are added. As described in the methodology section, we used an AMD/DDR-2 systems cerca 2009. In the future, we would like to repeat our experiments on newer multicore systems to see if that would change the results.

To validate our methodology of using pcap files, we also ran our tests using a separate machine to replay the pcaps onto the network at various speeds rather than reading the pcap file directly into the IDS under tests. These live traffic tests were done by replaying packets from the ICTF 2010 capture at the rate which they were originally captured at. The packets were rewritten to make use of the 10.10.1.0/24 network configuration that our systems were set to. Both

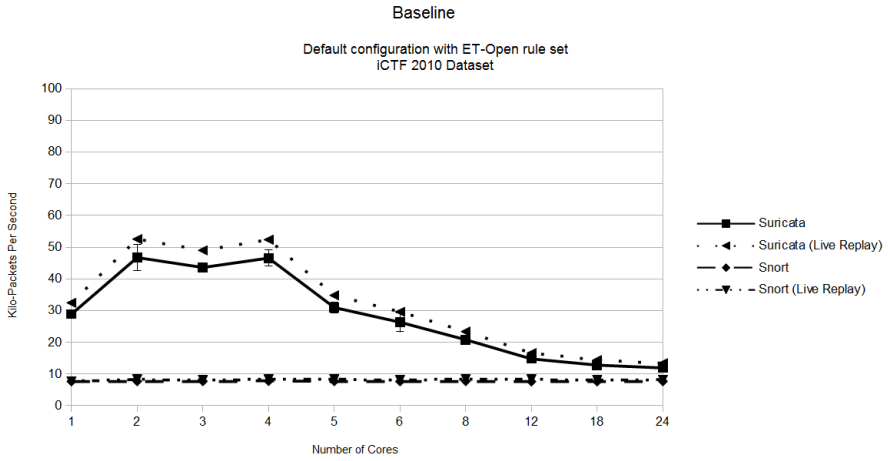

Fig. 1.

Snort and Suricata were set to monitor this IP range and our interface was set to promiscuous mode. As the tests show our maximum Suricata performance was around 95,000 PPS.[3] We attribute this jump in performance compared to Suricata running in PCAP read mode to issues we suspect exist with both read latency and disk I/O.

Overall, there is little difference in the results with live traffic replay. Reading the traces from the pcap eliminates the need for a separate replay machine and thus reduces the complexity of the experimental infrastructure. It is also interesting to note that when reading the pcap files directly there is no possibility of dropped packets because the system reads the input as fast as it can process it and no faster. This improves repeatability of the tests. In the remainder of the tests, we read packets directly from the pcap traces.

4.2 Varying the Rulesets

Figure 2 explores the impact of different rulesets using the same iCTF workload. In addition, to Snort and Suricata baseline measurements with the ET-Open ruleset that was shown in Figure 1, we add three additional lines - one for

[3] Many IDS Studies use PPS as the primary metric for measuring performance of a system. Some however, may prefer a metric of Gbps especially when deploying a system in a known network configuration. For our live tests we replayed traffic from a recorded dataset which had varying packet sizes. We set a fixed maximum transmission unit (MTU) size of 9600 Bytes for our 10GigE Interface. In the event that all packets being replayed were at the MTU of 9600 Bytes it would take approximately 139,000 PPS to reach 10 Gbps.

Suricata and one for Snort in which each one uses the ET-Pro Ruleset that is tuned specifically for its use and one for Snort with the VRT Ruleset that is also specifcally tuned for its use. Suricata sees some modest performance gain between 1 and 4 cores from the ET-Pro Ruleset, but overall our results show little performance difference in the use of one ruleset over the other. Once again, the performance of Snort is flat as additional cores are added and the performance of Suricata peaks between 2 and 4 cores.

It is worth noting that differences in ruleset can cause substantial difference in performance. System administrators are warned of the possible impact of adding poorly written custom rules. However, we saw little difference between the rulesets even though ET-Open rules are contributed by community members and ET-Pro rules are professionally tuned. The ET-Open set is still vetted by the Emerging Threats team and would presumably not contain any excessively expensive rules.

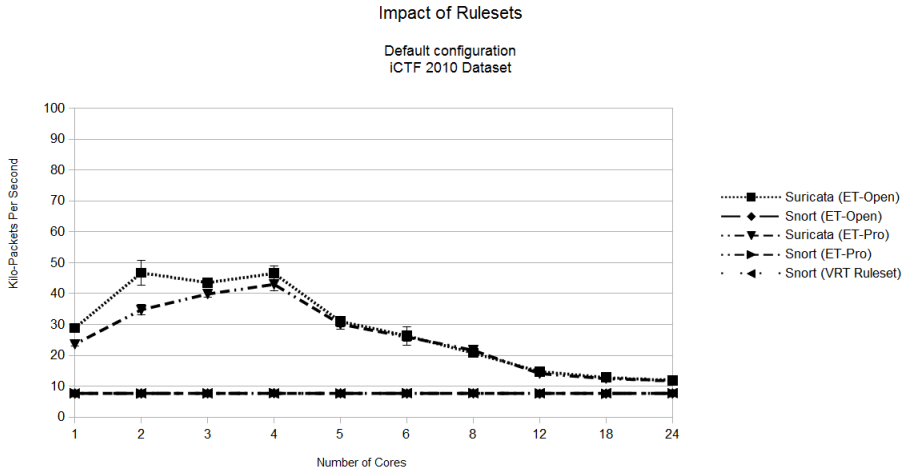

Fig. 2.

Figure 3 compares the Snort and Suricata baseline measurements to two additional lines showing the performance when no rules are loaded. Clearly this is an unrealistic configuration as no one would bother running an IDS without any IDS rules. However, it does shed some light on the scalability of the engine as the size of the ruleset changes. The difference in performance for Snort with no rules is substantial (26648 PPS (no rules) vs 7683 PPS (with ET-Open rule set)) or roughly 3.4 times faster then with a ruleset. Suricata also sees a performance difference at 1 and 2 cores, but the percentage difference is

substantially lower. Interesting, Suricata with different rulesets performs almost identically from 3 cores to 24 cores. It is possible that the same bottleneck that is reducing Suricata's performance as the number of cores increase is also hiding the performance impact of increasing ruleset size. This data suggests that Suricata is indeed realizing some additional scalability with ruleset size due to its multi-threaded approach.

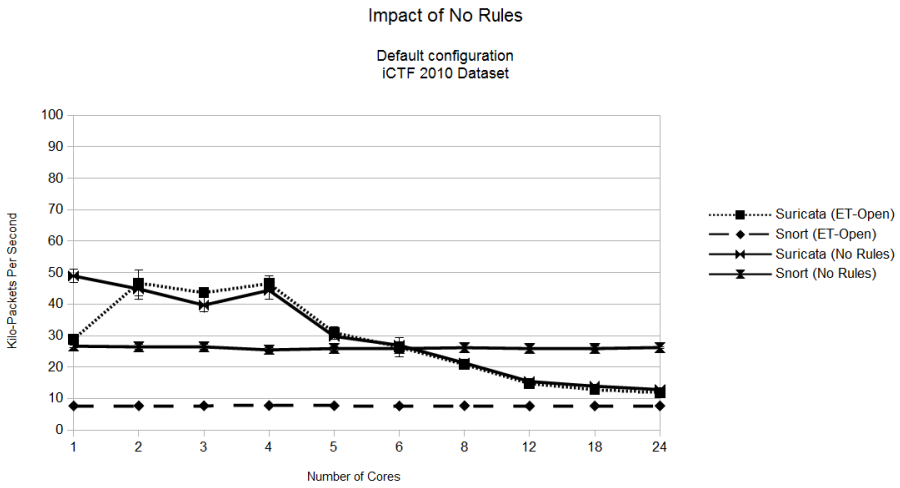

Fig. 3.

4.3 Varying the Configuration

Figure 4 explores the impact of configuration options on tuning the performance of an IDS. We compare our baseline Snort and Suricata measurements (default configuration) to the performance optmized configurations of each IDS as described in the methodology section (for Sucuricata, run-mode to autofp and max-pending-packets to 65535 and for Snort, ac mode and max-pkt-time to 1000). With these settings, there was a substanial performance increase in the number of PPS that Suricata is able to process (up to 6X). However, even with the tuning, there was still a substantial decrease in performance as cores were added. Snort showed little improvement from the performance optimized configuration.

In Figures 5, we shift from examining average packets per second to CPU utilization. The y-axis shows average CPU utilization. With 24 cores, the maximum value is 2400% or 24 cores are 100% utilized. The results of the using the ET-Open Ruleset with both Snort and Suricata in their default and performance

optimized configurations. Snort remained consistent at 100% across all configurations. Snort was limited to a single thread and was using 100% of the CPU on its assigned core. The default configurations of Suricata showed a gradual increase in CPU utilization as more cores were added. The performance opitimized configuration of Suricata is similar but shows a more pronounced leveling of CPU utilization between 4 and 12 cores.

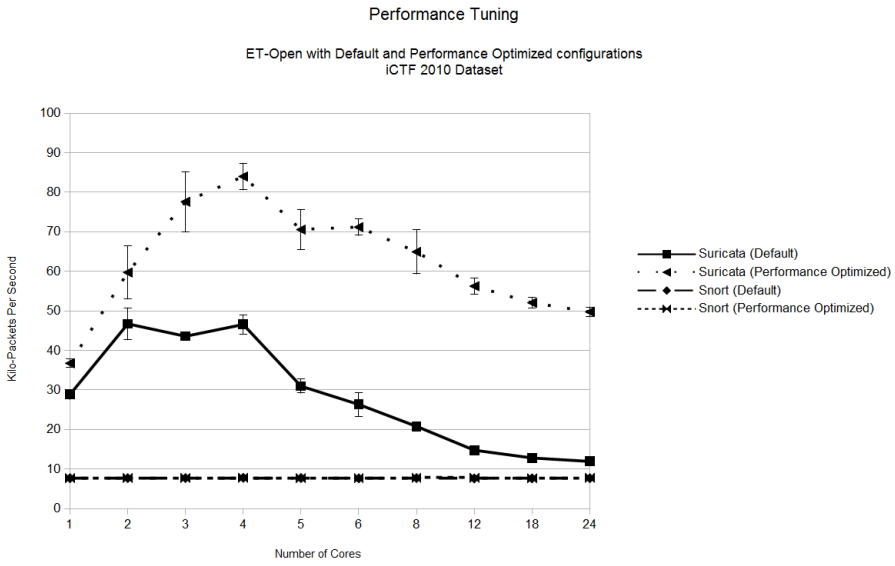

Fig. 4.

Average CPU utilization, and maximum CPU utilization are shown in Figure 5. Base Suricata, without performance optimization had a gradual increase in maximum CPU utilization as more cores were added, but never used more than 4 full cores (400%). Suricata, with performance optimization was able to use nearly all 24 cores (2400%) at times, but recall from Figure 5 that its average running 24 cores remained around 3 cores with full utilization (300%). This indicates that while Suricata is able to use additional CPU resources at times, there is little evidence to suggest an effective, common case use of parallelism as more and more cores are added. A dip in maximum CPU utilization from 8 to 12 cores is also shown.

We also took a look at the average memory usage for both Snort and Suricata. As in Figure 5, Figure 6 shows results using the ET-Open Ruleset with both Snort and Suricata in their default and performance optimized configurations.

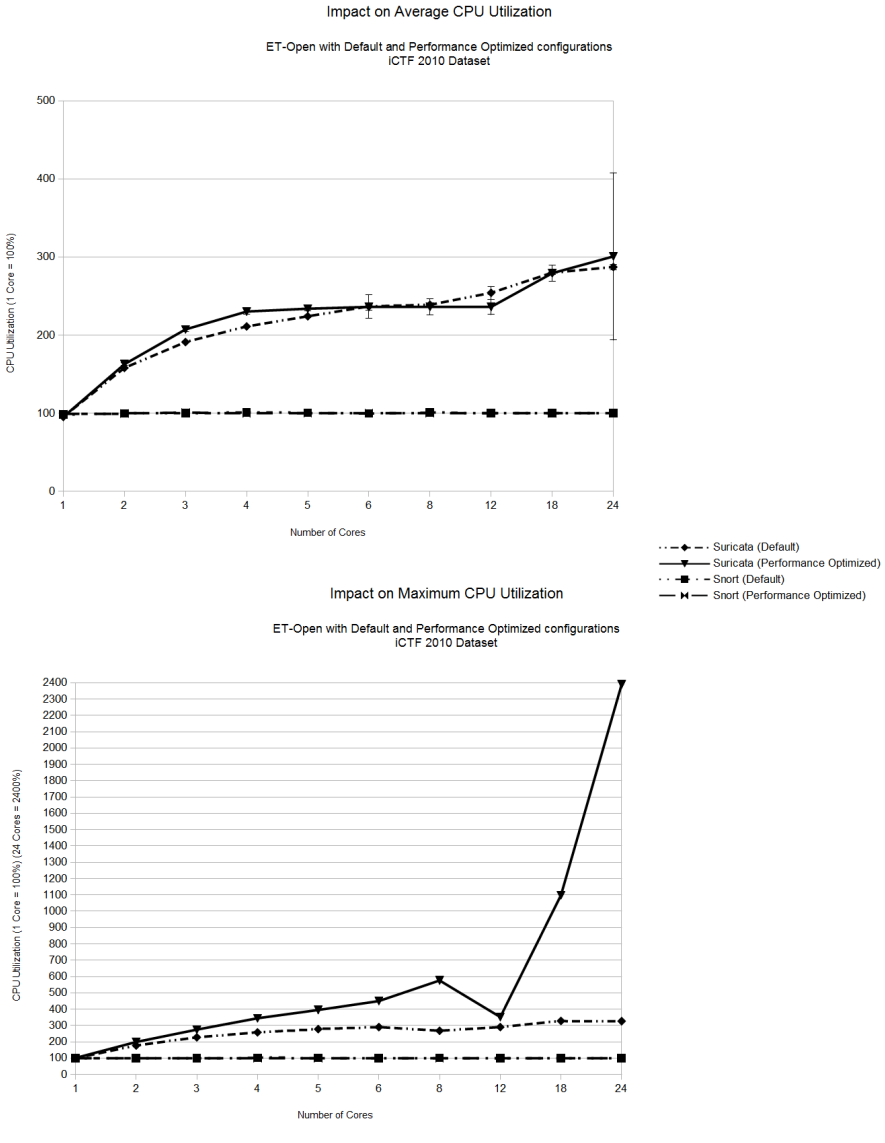
Impact on Average CPU Utilization
ET-Open with Default and Performance Optimized configurations
iCTF 2010 Dataset
CPU Utilization (1 Core = 100%)
Number of Cores
Suricata (Default)
Suricata (Performance Optimized)
Snort (Default)
Snort (Performance Optimized)
Impact on Maximum CPU Utilization
ET-Open with Default and Performance Optimized configurations
iCTF 2010 Dataset
CPU Utilization (1 Core = 100%) (24 Cores = 2400%)
Number of Cores

Fig. 5.

Snort used the least memory and not surprisingly, the amount of memory used did not vary with the number of available cores. The performance optimized configuration of Snort used substantially more memory as expected. Both the base and optimized configurations of Suricata show increasing memory usage as more cores are used. As with Snort, the optimized configuration of Suricata showed substantially more memory usage than the base configuration. Using more memory to obtain higher performance can be a good tradeoff. The machine we used had 48 GB of memory and on average our tests used less than 700 MB.

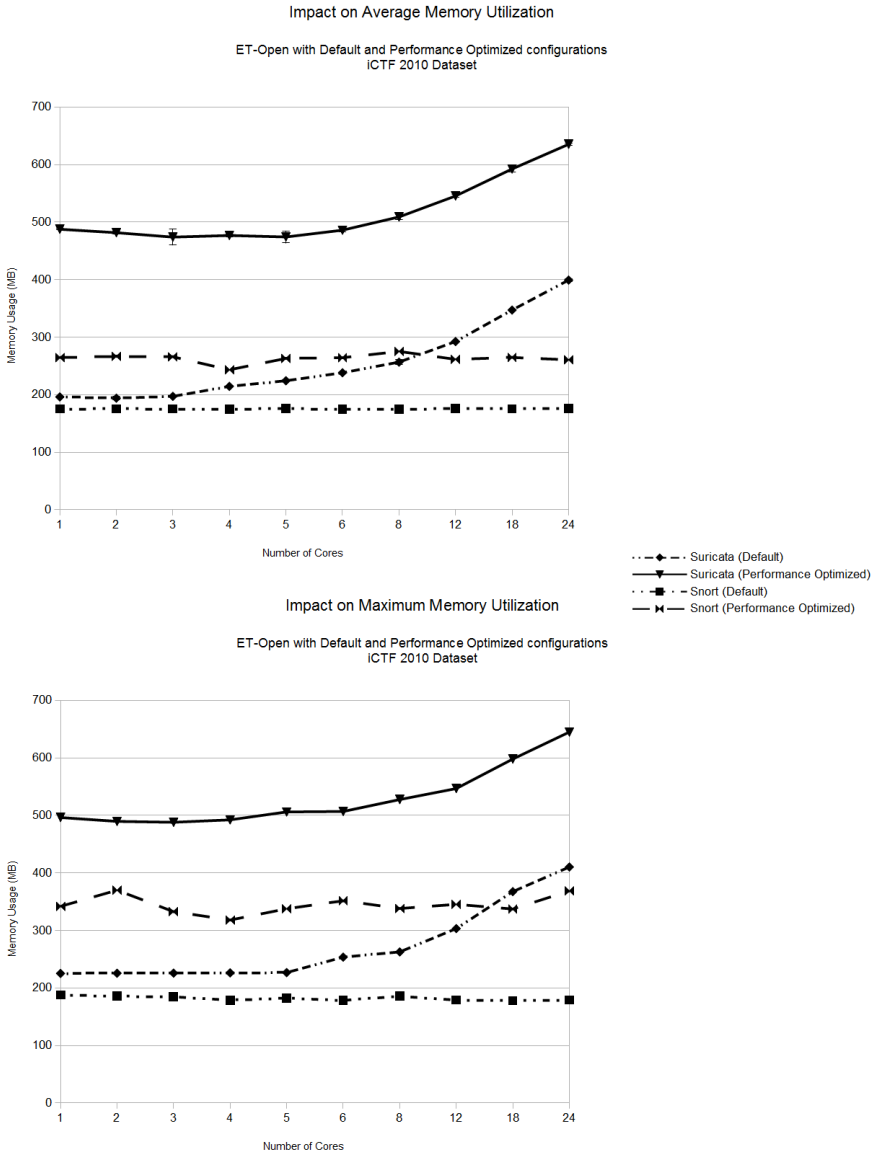
Impact on Average Memory Utilization
ET-Open with Default and Performance Optimized configurations
ICTF 2010 Dataset
Memory Usage (MB)
Number of Cores
Suricata (Default)
Suricata (Performance Optimized)
Snort (Default)
Snort (Performance Optimized)
Impact on Maximum Memory Utilization
ET-Open with Default and Performance Optimized configurations
ICTF 2010 Dataset
Memory Usage (MB)
Number of Cores

Fig. 6.

4.4 Varying the Workload

We experimented by different workloads and their effects on each IDS. For all previous figures, we used the iCTF pcap trace, but here we compare two runs with traces of individual Pytbull tests [9].

Figure 7 shows the results of running Snort with the ET-Open ruleset at 1 core and Suricata with the ET-Open ruleset across 1, 4, and 24 cores with different workloads. As we saw in our baseline results in Figure 1, these results demonstrate that Suricata's PPS throughput peaks at 4 cores on most workloads. This is most notable on the iCTF and Evasion Techniques workloads. The Evasion Techniques, included in Pytbull, represent a bundle of attacks, with slight modifications, made to evade an IDS. This can include obfuscating the payloads, protocol violations, overlapping packet fragments, and more. This set of techniques is made to challenge the IDS capabilities. Across almost every workload, Suricata has better performance with the exception of the Evastion Techniques workload at 1 and 24 cores.

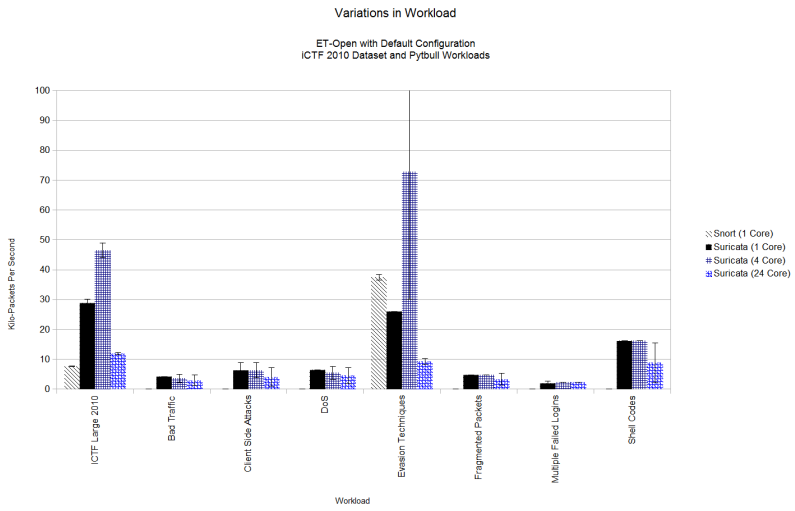

Fig. 7.

5 Conclusion

Our results show that Suricata outperforms Snort, even on a single core, where we expected Snort to have an advantage. However, there are problems with the scalability of Suricata as additional cores are added. Even though Suricata used up to a maximum of 24 full cores at some times, the average number of cores used, peaks around 3. We also saw a surprising drop in packets per second processing as more cores are added from 4 to 24 cores. We explored variations in rulesets, in configuration, and in workload, but our base conclusions held across these variations.

6 Future Work

The results we presented in this paper could be easily be augmented with tests on additional hardware platforms, with trace of additional workloads and using different configurations of Snort and Suricata. We plan to continue this work ourselves, but we are also hoping that other groups will take our testing infrastructure and use it in their own environments - with their own rulesets and workloads. We would like to found a repository of results to which others could contribute.

A community respository containing results of independent performance comparisons across many environments could help substantially in demystifying the choice of IDS for many organizations. We hear claims of substantially higher performance than we observed in our local tests, but when we ask questions about the configuration or details of the hardware required, few details are available. Snort considers this type of information on its products to be proprietary and it is therefore difficult to verify independently. Running Snort in a multi-process, parallel configuration on multiple cores requires speial preprocessing and post-processing glue code/scripts and the hardware necessary to achieve packet per second ratings in the GPPS range appears to require hardware priced in the range of tens to hundreds of thousands of dollars. [4] The main Snort web page mentions 400,000 registered Snort users and we wonder what percentage are running Snort in such customized configurations. One goal of our paper is to provide repeatable performance experiments in more standard/commodity environments.

[4] One concrete datapoint we recevied comes from Bivio. The Bivio 7500 Series appliance uses hardware stream splitting and hardware application load balancing with custom FPGA's to split Snort across 48 Cores (i.e. multiple instances of Snort). They report around 10 Gbps of throughput. This 48 core solution is 8U in size, consisting of 4 x 2U appliances ganged together, consumes 2400 Watts of power, and costs around $100,000 per appliance. This configuration is not a Snort certified solution, but rather a solution developed especially for the DOD community.

In the short term, we are especially interested in futher exploring the cause of Suricata's decrease in performance with more than 4 cores. We would also like to try running both Snort and Suricata on some of the newest multi-core systems to see how that impacts the results. Finally, we will continue to experiment with newer versions of both Snort and Suricata as they come out as well as additional tuning and configuration changes. We would welcome any suggestions from Snort developers, Suricata developers or the respective user communities on how to demonstrate the best possible performance of each IDS. As we identify successful strategies for achieving the highest performance on both Snort and Suricata for commodity platforms, we will update our website [21] with best practices.

7 Acknowledgements

We would like to thank Matt Jonkman, Victor Julien and Will Metcalf of the OISF for answering our questions happily and promptly every time. Additionally we would like to thank Eugene Albin for answering questions we had regarding his Graduate Thesis [13] on IDS testing. We would like to thank Bivio Networks for answering a number of questions about their products and how they run parallel instances of Snort. Finally we would like to thank a member of the Snort team that preferred to remain anonymous but whom answered questions regarding Snort configuration, optimization and preferred deployment.

References

[1]http://snort.org, December 2011.
[2]M. Roesch, "Snort: Lightweight Intrusion Detection for Networks", 13th Annual Systems Administration Conference (LISA), November 1999.
[3]"Snort: The De Facto Standard for Intrusion Detection and Prevention", http://www.sourcefire.com/security-technologies/open-source/snort, December 2011.
[4]SecTools.org, "Top 125 Security Tools", http://sectools.org, December 2011.
[5]Infoworld.com, "The Greatests Open Source Software of All Time", http://www.infoworld.com/d/open-source/greatest-open-source-software-all-time-776?source=fssr.
[6]J. Carr, "Snort: Open Source Network Intrusion Prevention", Esecurity Planet, http://www.esecurityplanet.com/prevention/article.php/3681296/Snort-Open-Source-Network- Intrusion-Prevention.htm, June 2007.
[7]J. Koziol, "Intrusion Detection with Snort", Sams Publishing, May 2003.
[8]Emerging Threats, http://emergingthreatspro.com, December 2011.
[9]Pytbull, http://pytbull.sourceforge.net, December 2011.
[10]Sysbench, http://sysbench.sourceforge.net/docs/, December 2011.
[11]S. Demaye, "Suricata-vs-snort", http://www.aldeid.com/wiki/Suricata-vs-snort, December 2011.
[12]OISF, http://www.openinfosecfoundation.org/projects/suricata/wiki, December 2011.

[13]E. Albin, "A Comparative Analysis of Snort And Suricata Intrusion Detection Systems", Naval Postgraduate School, Dudley Know Library, September 2011.
[14]iCTF pcap Dataset, "Full Packet Capture of (Attack Against Litya)", International Capture The Flag, http://ictf.cs.ucsb.edu/data/ictf_f2010/ictf2010pcap.tar.gz , December 2010.
[15]https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricatayaml, December 2011.
[16]http://lists.openinfosecfoundation.org/pipermail/oisf-users/2011-August/000820.html, December 2011.
[17]http://lists.openinfosecfoundation.org/pipermail/oisf-users/2011-February/000447.html, February 2011.
[18]http://lists.openinfosecfoundation.org/pipermail/oisf-users/2011-August/000820.html, August 2011.
[19]http://lists.openinfosecfoundation.org/pipermail/oisf-users/2011-December/001109.html, December 2011.
[20]http://www.bivio.net/public_pdfs/Bivio_7000_DS.pdf, December 2011.
[21]Snort and Suricata Performance Comparison, http://www.clarkson.edu/class/cs644/ids/, December 2011.
[22]B. Whaley, "Snort, Suricata creators in heated debate Are the open source projects irrevocably damaged?", Network World US, July 2010.
[23]J. Vijayan, "DHS vendors unveil open source intrusion detection engine", Computerworld, July 2010.
[24]E. Messmer, "Is open source Snort dead? Depends who you ask", http://www.networkworld.com/news/2010/072010-is-snort-dead.html, NetworkWorld, July 2010.
[25]M. Jonkman, Personal Email Communication, February 13 2010.
[26]Valimaki, Mikko., "Dual Licensing in Open Source Software Industry," Aalto University, Systemes d'Information et Management, Vol. 8, No. 1, pp. 63-75, 2003,
[27]Department of Homeland Security Host Program, "Suricata as an Exemplary example of DHS innovation", http://www.cyber.st.dhs.gov/host/
[28]Snort, VRT Subscription options, http://www.snort.org/vrt/buy-a-subscription/, February 2012.
[29]GameLinux.org, "Some Notes on Making Snort Go Fast Under Linux", http://www.gamelinux.org/ page id 284, December 2011.
[30]http://lists.emergingthreats.net/pipermail/emerging-sigs/2011-January/011641.html, January 2011.
[31]Mikelococo.com, "Capacity Planning for Snort IDS", http://mikelococo.com/2011/08/snort-capacity-planning/, August 2011.
[32]Snort.org, "Snort 2.9.2 Manual, Section 2.1", http://manual.snort.org/node16.html, 2011.
[33]Snort.org, "Snort 2.9.2 Manual, Section 2.5", http://manual.snort.org/node20.html, 2011.
[34]http://seclists.org/snort/2011/q2/32, April 2011.
[35]S. Sturges, "Using Perfmon and Performance Profiling to Tune Snort Preprocessors and Rules", http://www.snort.org/assets/163/WhitePaper_Snort_PerformanceTuning_2009.pdf, November 2009.