A Performance Analysis of Snort and Suricata Network Intrusion Detection and Prevention Engines

David J. Day
School of Computing and Mathematics
University of Derby
Derby, UK
d.day@derby.ac.uk

Benjamin M. Burns
School of Computing and Mathematics
University of Derby
Derby, UK
benburns01@googlemail.com

Abstract - **Recently, there has been shift to multi-core processors and consequently multithreaded application design. Multithreaded Network Intrusion Detection and Prevention Systems (NIDPS) are now being considered. Suricata is a multithreaded open source NIDPS, being developed via the Open Information Security Forum (OISF). It is increasing in popularity, as it free to use under the General Public Licence (GPL), with open source code. This paper describes an experiment, comprising of a series of innovative tests to establish whether Suricata shows an increase in accuracy and system performance over the de facto standard, single threaded NIDPS Snort. Results indicate that Snort has a lower system overhead than Suricata and this translates to fewer false negatives utilising a single core, stressed environment. However, Suricata is shown to be more accurate in environments where multi-cores are available. Suricata is shown to be scalable through increased performance when running on four cores; however, even when running on four cores its ability to process a 2Mb pcap file is still less than Snort. In this regard, there is no benefit to utilising multi-cores when running a single instance of Snort.**

Keywords – snort; suricata; performance; NIDS; NIDPS; multithreaded; multi-core; comparison; experiment

I. INTRODUCTION

Nielsen's Law states that the bandwidth available to users increases by 50% annually [1]. This exponential growth perpetrates design challenges for developers of Network Intrusion Detection and Prevention Systems (NIDPS). Once traffic levels exceed operational boundaries, packets are dropped and the system becomes ineffective. With pattern matching taking up to 70% of the total processing time [2];[3], copious research is focused on reducing the pattern matching overhead with inventive algorithms. Alternatively, some NIDPSs utilise specialist hardware such as Application Specific Integrated Circuits (ASICS) and Field Programmable Gateway Arrays (FPGA) providing parallelism to increase throughput [4]. However, these systems are costly, leaving some organisations restricted to using single threaded PC-based freeware, such as Snort.

With Internet bandwidth accelerating and Central Processing Unit (CPU) core speeds reaching a plateau, it is unlikely that a single threaded solution will remain effective. The relative influence of Moore's Law [5] on single threaded application performance is reducing and this is responsible for a developmental shift toward increasing power-density for multithreaded processing [6]. Consequently, almost all PC CPUs are now multi-core. However, multi-core processors are only as valuable as the multithreading software utilising them and Snort is not multithreaded.

To address this, Suricata has been released by the Open Information Security Foundation (OISF). It is an open source NIDS promising multi-threading and graphics card acceleration in the form of CUDA (Computer Unified Device Architecture) and OpenCL [7]. Other feature benefits include: Gzip Decompression, Automatic Protocol Detection, Flow Variables, Independent HTP library and Fast IP (Internet Protocol) Matching [8]. If Suricata delivers on the promises of the OISF it may meet the demands caused through exponential increases in network traffic.

This paper describes an evaluation of Suricata through critical comparison of both Suricata and Snort NIDPSs. The remainder of this paper is organised as follows: Section II describes the experiment design including empirical metrics, test-bed development, system stressing, traffic generation and attack methods. In Section III, the experiments are described and the results are reported. In Section IV, we present an analysis of the results, and finally, in Section V, we offer our conclusions.

II. EXPERIMENT DESIGN

A. Metrics

Antonatos et al. [3] suggest that the metrics to be used for measuring the performance of an NIDPS should be the attack detection rate, false positives and capacity. Limitations in capacity imply false negatives; once a NIDPS exceeds its capacity, packets will be dropped and subsequently any malicious content within them will not be detected. Mell et al. [9] define the quantitative metrics used for evaluating NIDPS accuracy as follows: coverage (amount of attacks possible to detect), probability of false alarm, probability of detection, attack resistance, ability to handle high bandwidth traffic and capacity. With regard to

 ISBN: 978-1-61208-116-8

capacity, it has a number of constituent components and thus, it is not a single metric. Table 1, informed by Hall & Wiley [10], illustrates some of the metrics that constitute capacity.

The above research informed that the following capacity metrics should be recorded: Bytes per second, packets per second and quantity of network attacks. In addition, for each NIDPS, the number of packets dropped, true positives, true negatives, false negatives, and the total amount of alarms were also recorded. Finally, the host resources monitored were, CPU and memory utilisation, persistent storage, network interface bandwidth and page file statistics.

TABLE 1 METRICS OF CAPACITY

Test Metrics	Resources Used
Packets per Second	CPU Cycles, network interface bandwidth, memory bus bandwidth.
Bytes per second (average packet size)	CPU Cycles, network interface bandwidth, memory bus bandwidth.
Protocol Mix	CPU cycles and memory bus bandwidth.
Number of unique hosts	Memory size, CPU cycles, memory bus bandwidth.
Number of new connections per second	CPU cycles and memory bus bandwidth.
Number of concurrent connections	Memory size, CPU cycles, memory bus bandwidth.
Alarms per second	Memory size, CPU cycles, memory bus bandwidth.

B. Test-bed

The test-bed was setup in a virtual environment, facilitating experiment portability and security. It also allowed for faster experiment initialisation. This was necessary for frequent repetition and re-configuration of the experiment tests.

Vmware workstation 6.5 was used as the virtualisation platform, largely due to superior IO and disk performance over competitors Virtual Box and Virtual PC [11]. Ubuntu 10.4 TLS 32 bit was chosen as the operating system. Ubuntu is frequently updated and has a good community base. Further it is the most popular Linux operating system [12]

The default NIDPS hardware configuration was a 2.8GHz (E5462) Quad-Core Intel Xeon, running with 1-4 cores and 3GB of DDR2 800MHz fully-buffered memory. Each system also had a maximum hard-drive capacity of 20GB. The network traffic was replayed for each system separately. The system used to replay the network traffic utilises a single core, as well as 1 GB of memory. The VMware host operating system utilised 2GB of memory and 1 core preventing the host from having any performance impacts on the test-bed.

Snort and Suricata were configured to run using identical rule-sets. Suricata uses a different classification configuration to Snort, which uses 134 decoder and 174 pre-processor rules. Both NIDPSs were using identical logging methods, namely, Barnyard, MYSQL and AcidBase. The versions of Snort and Suricata used were v2.8.5.2 and v1.0.2 respectively. Both systems used the Snort v2.8.5.2 VRT rule set, combined with the Emerging Threats rule set. After all rules were loaded, Suricata had 11039 detection rules loaded against Snorts 11065. This discrepancy was due to Suricata's failure to parse certain VRT rules.

C. Traffic

There are a number of considerations when choosing network traffic for NIDPS testing. Firstly, attack traffic can be used, either on its own, or, with the added context of background traffic. When using background traffic, this can either be real or simulated. If it is real, it could be left intact or alternatively, sanitised [9] i.e. payload and ip address information removed.

For the test to be useful, it is deemed desirable to use real network background traffic. However, repetition of the experiments, using real-time network traffic, would be unpredictable due to its dynamic nature. Our solution was to use traffic that had been captured to a pcap (packet capture) file. This facilitated their processing by the NIDPSs in offline mode, allowing for replay on the network at different speeds, using TCPReplay [13]. Further, any risk to mission critical networks was removed.

There are numerous test traffic sources available online for download, unfortunately, these are often sanitised. This renders them useless for evaluating content matching NIDPS, which perform deep packet inspection. Tools do exist which can add random payloads into sanitised data, e.g., TCPdump Randomiser [14], however, the realism of such modified data becomes questionable. Hacking contests also offer sources of traffic capture, although the traffic content is not documented, hence this must be predetermined prior to use, e.g., which attacks were used and which were successful. As a result of these issues, it was decided to capture background traffic from a busy universities web and application server. This was then merged with exploit traffic, created using the Metasploit Framework [15]. The Metasploit Framework contains a total of 587 exploit modules [15], allowing attack data to be easily generated in quantity.

The exploit traffic was captured by performing attacks via Metasploit to a Microsoft Windows 2000 machine. Windows 2000 was chosen as there are more Metasploit exploits for this operating system than any other. Numerous services and discontinued applications where installed to facilitate as many of these attacks as possible. Regrettably, they could not all be obtained. The attacks perpetrated are

 ISBN: 978-1-61208-116-8

shown in Table 2, captured using Wireshark [16]. With the background and attack traffic captured, the two were combined. Part of the Wireshark application, Edicap, was used to modify the timestamp of the exploit traffic, to correlate with the background traffic. With this done, the two were merged together in chronological order, such that the attack traffic was distributed within the background traffic.

D. Stressing the system

The capacity of a NIDPS is closely connected to the CPU capacity of the system [2]. Thus, Snort and Suricata should be subjected to CPU impairment, to evaluate their efficacy under stressful conditions.

TABLE 2 EXPLOITS PERFORMED

Code	Name	Description
ms03_026_dcom	Microsoft RPC DCOM Interface Overflow	Module exploits a stack buffer overflow in the RPCSS service
ms05_039_pnp	Microsoft Plug and Play Service Overflow	Stack buffer overflow in the Windows Plug and Play service
ms05_047_pnp	Microsoft Plug and Play Service Registry Overflow	Stack buffer overflow in Windows PnP services. Causes Reboot
ms06_040_netapi	Microsoft Server Service NetpwPathCanonicalize Overflow	Stack buffer overflow in the NetApi32 CanonicalizePathName() function using the NetpwPathCanonicalize RPC call in the Server Service
ms05_017_msmq	Microsoft Message Queueing Service Path Overflow	Exploits a stack buffer overflow in the RPC interface to the Microsoft Message Queueing service
ms01_033_idq	Microsoft IIS 5.0 IDQ Path Overflow	exploits a stack buffer overflow in the IDQ ISAPI handler for Microsoft Index Server

VMware was used to allow the number of logical and physical cores to be reduced. The cores themselves were stressed by generating threads, causing an adjustable and measureable workload. This was performed using the application cpulimit [17], which generates configurable workloads across the processor, allowing for the total amount of stress applied by each thread, to be limited by a percentage of the CPU capacity.

Snort and Suricata both provide the ability to replay pcap files internally. This is done at the maximum speed possible for the NIDPS, providing a good metric as to the performance of a system. Yet, using this method the maximum loss free rate (MLFR) cannot be accounted for. Therefore, TCPReplay [13] was used to control the traffic replay rate, thereby allowing for stress testing under network load.

E. System Monitoring

The following resources were monitored: CPU utilisation, memory utilisation, persistent storage bandwidth and network interface bandwidth. This was performed using the Linux command line utility dstat.

F. Experiment protocol

Throughput speeds are increasing [18], and the MLFR of NIDPSs is affected by both the utilisation of the CPU and the throughput of the traffic [10]; [3]. Thus, the experiment was designed to provide data regarding how each system performs, with increased throughput and under increased CPU stress.

Attack traffic was played to both NIDPSs, with varying CPU configurations. These were: core configuration of 2 processing cores, 1 core, 50% and 75% load. The ability for the NIDPSs to read the packets, along with the accuracy of alerts, was measured, with special attention being paid to the false negative rate. The test traffic was replayed into the environment through TCPReplay, at a multiplier of 40, i.e., replayed 40 times faster than it was captured. This results in a reported playback throughput of 3.1 Mbps, and a packet drop rate of under 2%. This ensured the experiments could be completed in a timely fashion, on the threshold of packet loss.

Each time a test was run, the start and end times of the NIDPS start-up and traffic replay were recorded. This provided a good reference point when analysing the alerts and system statistics. For each test run, the alert output information was recorded using acidbase, as well as the unified2 alert output file being archived for future reference. Statistics produced on NIDPS close down, reported, the number of generated alerts, how many packets were processed, and what ratio of network protocols were handled. Any traffic travelling from hosts 192.168.16.2 and 192.168.16.128 was known to be malicious traffic.

III. RESULTS

This section reports and analyses the results for each NIDPS, for accuracy, dropped packet rate, system utilisation and offline speed. Each of these is now discussed in turn.

A. Accuracy

To determine accuracy, control alerts were used. These are alerts generated without system stress, used as a

 ISBN: 978-1-61208-116-8

baseline. Deviation from the baseline under stress is an indication of a change in detection accuracy.

Table 3 shows the number of alert types generated when the attacks were performed against each NIDPS. Figure 1 shows Suricata alerted on every exploit, under all configurations, yet some alerts types were lost, resulting in a reduction of detection breadth [19].

TABLE 3 ALERTS GENERATED BY SNORT AND SURICATA

Alert	Snort	Suricata
ms05_040_pnp	4	4
ms05_047_pnp	1	1
ms05_039_pnp	1	6
ms03_026_dcom	1	2
ms01_033_1dq	2	4
ms05_017_msmq	2	3

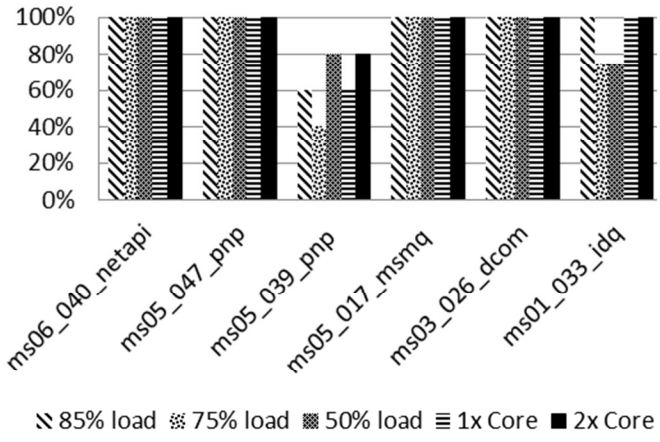

Figure 1 Suricata Alerts

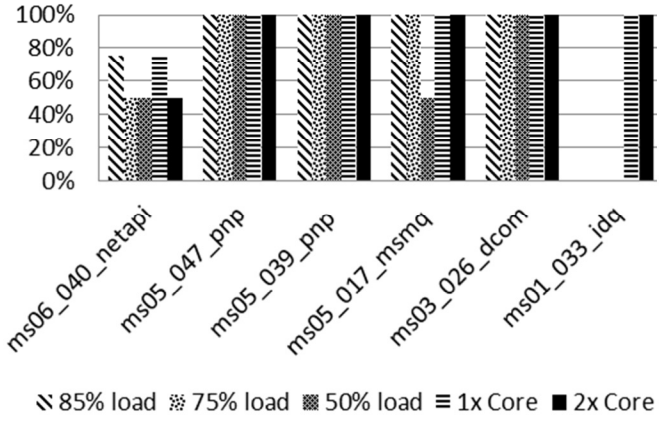

Figure 2 Snort Alerts

Figure 2 shows Snort fails to alert on ms01_033_idq. This is a false negative caused by excessive load.

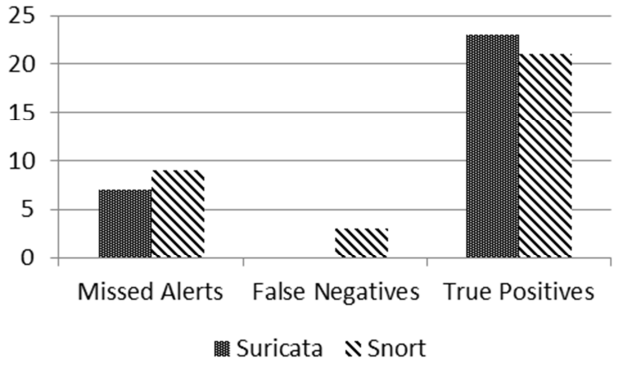

Figure 3 Attack accuracy measurements

Figure 3 shows the number of false positives (fp) and true positives (tp) for both NIDPSs, relative to the number of missed alerts by each system.

B. Dropped Rate

False negatives (fn) can be caused by dropped packets. Figure 4 plots the amount of packets dropped by Snort and Suricata as the CPU availability drops. While Snorts percentage drop is largely linear, Suricata's performance diminishes significantly once the CPU availability reduces below one core. Figure 5 shows how reducing the number of cores, and stressing the CPU, effects false negatives on both systems.

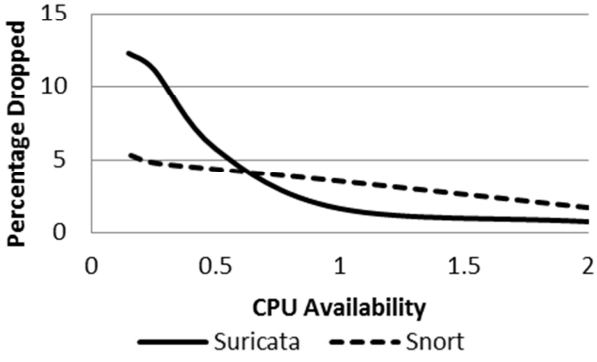

Figure 4 Packet loss at 3.2 MBps

C. System utilisation

Figure 6 shows the relationship between CPU utilisation, and network throughput, on both Suricata and Snort. It depicts how CPU load increases relative to network throughput. This behaviour is more prominent whilst running Suricata, with Snort exhibited similar behaviour on much smaller scale.

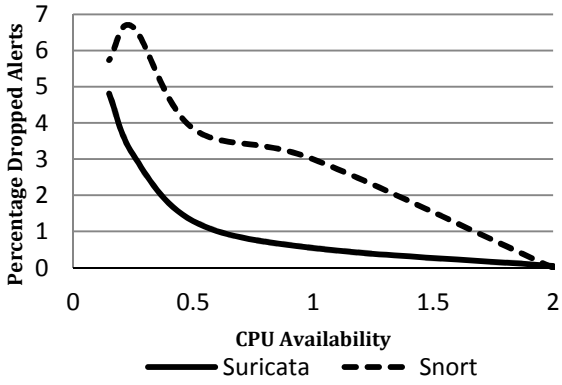

Figure 5 False Negatives (dropped alerts)

With dual-cores available, Suricata has a lower drop rate than Snort. To investigate why, both systems were evaluated for their ability to utilise both cores. Figures 7 and 8 show how Snort and Suricata (respectively), utilise dual core processors.

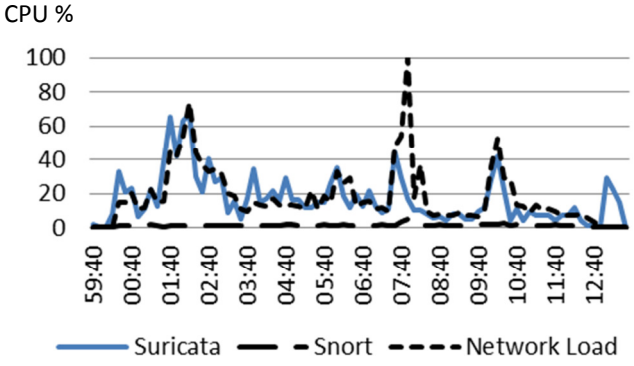

Figure 6 Network throughput and CPU utilisation for the Single Core Configuration

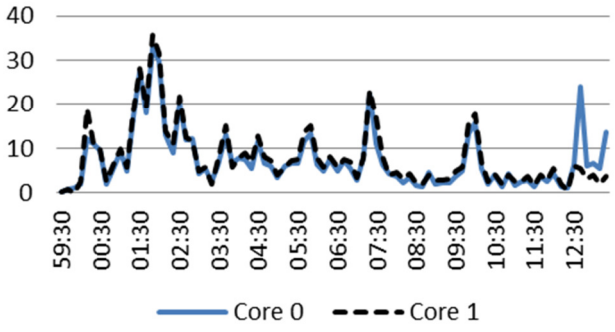

Figure 7 Suricata utilising dual cores

Figure 7 shows that Suricata utilises the 2 cores uniformly, compared to Snorts more erratic load balancing, Figure 8. This is consistent with expectations due to Suricata's multithreaded design.

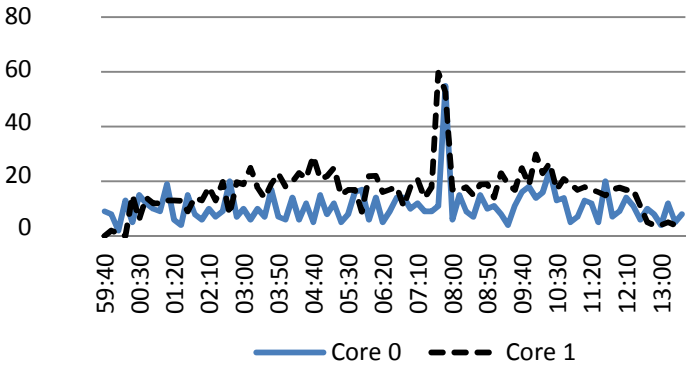

Figure 8 Snort utilising dual cores

Both NIDPSs have the ability to process traffic in offline mode by receiving a pcap file and processing it at maximum capacity. This was performed to identify the speed in which both systems can process traffic. The test was performed for both NIDPSs, using the same pcap file. The time each system took to process the file is displayed in Figure 12.

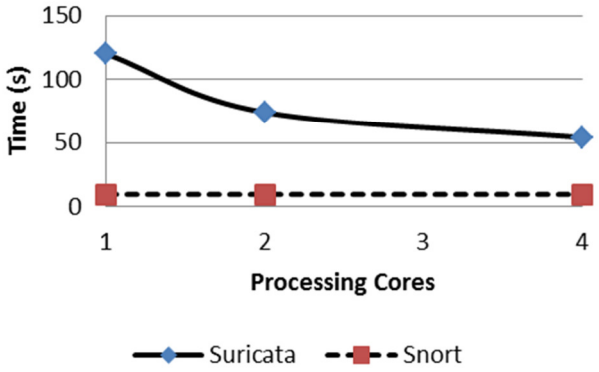

Figure 9 Pcap processing time (offline mode)

Additional cores did not improve Snorts processing time, although Suicata's performance increased by 220%, when using four cores, compared to one. Again this is expected, considering Suricata's multithreaded design.

IV Result Analysis

Arguably the most important metric to evaluate NIDPSs, is accuracy. This has been described as the system's attack coverage, false positives, false negatives, capacity, and ability to handle high bandwidth traffic [9]. The experiments outlined provided details regarding all of these.

The developers of Suricata have stated that their primary focus is improving NIDPS accuracy [20]. With Suricata having a higher accuracy than Snort, our experiments show that they have had some success. This is evident in Figures 1,2 and 3, including data showing that Snort failed to alert on the ms01_033_idq exploit with the processor loaded at 50% or above. A partial reason is Snort having less control alerts triggered by the attack than Suricata (two compared

 ISBN: 978-1-61208-116-8

with four). Snort failed to alert on ms01_033_idq using two rules from the VRT rules set, i.e., ID 1245 and 1244. Suricata succeeded in that these same alerts were triggered.

Larger processing requirements demanded by Suricata caused it to reach its operational capacity quicker than Snort, explaining the greater number of dropped packets under stress. By comparison, Snort places less demand on the system, enabling it to have a reduced packet drop rate at peak system loads. Figure 4 shows the percentage of dropped packets increasing steeply, once CPU availability is reduced to a single stressed core. The proportional relationship between dropped packets, and false negatives, is demonstrated for both systems in Figure 5.

When Suricata is run on a multi-core configuration it has a lower packet loss rate than Snort. Figures 7 and 8 show that Suricata uses available cores, on a dual core system, in a more uniform fashion. Offline tests show that Suricata was considerably slower than Snort. Alhough multiple cores relates to a more marked improvement with Suricata, than Snort, see Figures 4,5 and 9. In this sense, it could be argued that, Suricata possesses an improved ability to provide scalability. Nevertheless, in circumstances when the bandwidth received is greater than Snort can handle, the recommendation is to run multiple instances of Snort on multiple processor cores [21]. This could provide scalability similar to that of Suricata, albeit with added cost of processing a single threaded application over multiple cores.

V. Conclusions

The analysis of the results has shown that Suricata has a higher accuracy rate than Snort, although this comes at the cost of putting an increased relative demand on the CPU. The results show that, due to utilising multiple cores more uniformly, Suricata has the potential to be a more scalable and efficient, where multiple cores are available. However, due to the higher resource demands of Suricata, the accuracy would be expected to diminish, when used in low commodity, single core, deployment.

This research has endeavoured to classify the performance benefits of the innovative Suricata engine. Whilst the potential of Suricata is significant, at present, its development is incomplete. Since acceptable performance is not guaranteed, trial implementations of the engine would be advised. This would provide an opportunity for feedback; accelerating the developmental process of this pivotal detection and prevention engine. In addition to this, future research should pay attention to documenting Suricata's performance, whilst utilising even larger numbers of cores.

REFERENCES

[1] J. Nielsen, "Nielsen's Law of Internet Bandwidth," *useit.com: Jakon Nielsen's Website,* [Online] 5 April 1998, [Cited: 4 January 2011.] http://www.useit.com/alertbox/980405.html.

[2] J.B.D. Cabrera, J.Gosar, and R.K. Mehra, "On the statistical distribution of processing times in network intrusion detection," 43rd IEEE Conference on Decision and Control, vol. 1, IEEE Press, 2004, pp. 75-80, doi: 10.1109/CDC.2004.1428609.

[3] S.Antonatos, K.Anagnostakis, and E. Markatos, "Generating realistic workloads for network intrusion detection systems," Proceedings of the 4th ACM workshop on software and performance, ACM, 2004, pp. 207-215, doi: 10.1145/974043.974078

[4] M. Abishek, W. Najjar, and L.Bhuyan, "Compiling PCRE to FPGA for accelerating SNORT IDS," ACM, 2007, Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems , pp. 127-136. doi: 10.1145/1323548.1323571.

[5] G. Moore, "Cramming more components onto integrated circuits," Electronics, McGraw Hill, Vol. 38, Num. 8, 19 April 1965

[6] A. Ghuloum, "Face the inevitable, embrace parallelism," Communications, vol. 52, ACM, September 2009, pp. 36-38. doi: 10.1145/1562164.1562179.

[7] M. Jonkman, "Suricata IDS Available for Download," *Seclists.org*, [Online] 2009, [Cited: 12 May 2010.] http://seclists.org/snort/2009/q4/599.

[8] OISF, "The open information security foundation," [Online] 2010, [Cited: 4 October 2010.] http://www.openinfosecfoundation.org/index.php?start=15.

[9] P. Mell, V. Hu, R. Lippmann, J. Haines, and M. Zissman, "An Overview of Issues in Testing Intrusion Detection Systems," [Online], [Cited: 16 September 2010.] http://csrc.nist.gov/publications/nistir/nistir-7007.pdf.

[10] M. Hall, and K. Wiley, "Capacity Verification for High Speed Network Intrusion Detection Systems," Lecture notes in computer science, Springer, 2002, vol. 2516, pp.239-251. doi: 10.1007/3-540-36084-0_13.

[11] P. Domingues, F. Araujo, and L. Silva, "Evaluating the Performance and Intrusiveness of Virtual Machines for Desktop Grid Computing," Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing, IEEE, 2009, pp. 1-8. doi: 10.1109/IPDPS.2009.5161134.

[12] L. Bodnar, "Page hit ranking," *DistroWatch.com,* [Online] 2010, [Cited: 20 April 2010.] http://distrowatch.com/.

[13] A. Turner, "TCPReplay pcap editing & replay tools for *NIX," *TCPRepla,* [Online] 23 August 2010, [Cited: 13 December 2010.] http://tcpreplay.synfin.net/.

[14] Institure of Computer Science, "Network monitoring for security: intrusion detection systems*" Institure of Computer Science*, [Online] 6 August 2007, [Cited: 12 December 2010.], http://dcs.ics.forth.gr/dcs/Activities/Projects/ids.html.

[15] Rapid 7, "Metasploit – penetration testing resources," *Metasploit,* [Online] 2010, [Cited: 1 October 2010.] http://www.metasploit.com/.

[16] Wireshark.org.uk, "Wireshark," *Wireshark.org.uk,* [Online] [Cited: 14 April 2010.] http://www.wireshark.org/.

[17] A. Marletta, "CPU usage limiter for Linux," *Sourceforge.net,* [Online] 29 November 2010, [Cited: 13 December 2010.] http://cpulimit.sourceforge.net/.

[18] M. Cloppert, "Detection, Bandwidth, and Moore's Law," *SANS Computer Forensic Investigations and Incident Response Blog,* [Online] 05 Jan 2010, [Cited: 05 May 2010.] https://blogs.sans.org/computer-forensics/2010/01/05/.

[19] C. Jordan, "Writing detection signatures," USENIX, December 2005, ;login, vol. 30, pp. 55-61.

[20] V. Julien, "On Suricata Performance," Inliniac [Online] 2010, [Cited: 06 October 2010.] http://www.inliniac.net/blog/2010/07/22/on-suricata-performance.html.

[21] N. Houghton, "Single Threaded Data Processing Pipelines and Intel Achitectures," *VRT*, [Online] Vulnerability Research Team, 7 June 2010, [Cited: 2010 12 17.], http://vrt-sourcefire.blogspot.com/2010/06/single-threaded-data-processing.html.

 ISBN: 978-1-61208-116-8