2012 26th International Conference on Advanced Information Networking and Applications Workshops

A Realistic Experimental Comparison of the Suricata and Snort Intrusion-Detection Systems

Eugene Albin and Neil C. Rowe
Dept. of Computer Science
U.S. Naval Postgraduate School
Monterey, California, United States
ncrowe@nps.edu

***Abstract*—The Suricata intrusion-detection system for computer-network monitoring has been advanced as an open-source improvement on the popular Snort system that has been available for over a decade. Suricata includes multi-threading to improve processing speed beyond Snort. Previous work comparing the two products has not used a real-world setting. We did this and evaluated the speed, memory requirements, and accuracy of the detection engines in three kinds of experiments: (1) on the full traffic of our school as observed on its "backbone" in real time; (2) on a supercomputer with packets recorded from the backbone; and (3) in response to malicious packets sent by a red-teaming product. We used the same set of rules for both products with a few small exceptions where capabilities were missing. We conclude that Suricata can handle larger volumes of traffic than Snort with similar accuracy, and that its performance scaled roughly linearly with the number of processors up to 48. We observed no significant speed or accuracy advantage of Suricata over Snort in its current state, but it is still being developed. Our methodology should be useful for comparing other intrusion-detection products.**

Keywords-intrusion detection, computer networks, Snort, Suricata, performance

I. INTRODUCTION

Network-security monitoring requires intrusion-detection and intrusion-prevention technologies within a defense-in-depth strategy [1, 2]. Numerous products are available. The open-source product Snort (www.snort.org) has become the de-facto industry standard for signature-based network intrusion-detection engines [3]. In 2009 the Open Information Security Foundation (OISF) released a new signature-based network intrusion-detection engine called Suricata (www.openinfosecfoundation.org). Suricata is an open-source engine envisioned to be the "next generation intrusion-detection system / IPS engine". Suricata has native multi-threaded operations, a feature useful as network bandwidth increases. Suricata also improves on Snort in state-based analysis, something particularly important for HTTP traffic. The typical Snort installation can process network traffic at a rate of 100-200 megabits per second before reaching the processing limit of a single CPU and needing to drop packets to compensate [4], and many networks today approach or exceed that limit. So Suricata appears poised to be a popular product with many implementations given its open-source character. A careful evaluation of it now is important.

II. PREVIOUS WORK

Intrusion detection is difficult to accomplish perfectly [5]. Faulty rule design is rare because most useful rules are quite simple. But rules may be imprecise because there is no distinctive signature for a particular kind of attack. False positives and false negatives can also occur through performance problems with the detection engine itself. If the hardware fails or becomes otherwise overused, it can drop packets and allow malicious ones onto a network. These issues make testing of intrusion-detection systems challenging [6].

Previous work compared the performance of Snort version 2.8.5.2 to Suricata version 1.0.2 [7]. Their testbed consisted of an Ubuntu virtual machine hosted in a VMWare Workstation virtual environment running on a 2.8GHz Quad-Core Intel Xeon CPU with 3GB of RAM. They examined detection engine speed as well as the accuracy under varying degrees of network and processor use. Alert generation was stimulated by injecting six known malicious exploits generated using the Metasploit framework (www.metasploit.com). The results showed that Snort was more efficient with system resources than Suricata, and when operating in a multi-CPU environment, Suricata was more accurate due to fewer false negative alerts. However they concluded that the overall performance of Suricata in a four-core environment was slower than that of Snort in a single-core environment

978-0-7695-4652-0 2012

DOI 10.1109/WAINA.2012.29

when processing 2 gigabytes of previously captured network data.

Another report compared the accuracy of Snort and Suricata in detecting a wide variety of malicious files and suspicious actions [8]. Creating a custom application in Python specifically designed to send a variety of specific vulnerabilities through an intrusion-detection system via a number of different vectors, this work attempted to measure the accuracy of both detection engines. The results showed that the two rule sets (VRT and ET) worked well together but required tuning to be most effective.

Another researcher tested the performance of the multi-threading capabilities of Suricata [9]. By adjusting the detect_thread_ratio and the cpu_affinity variables in the Suricata configuration file on a dual 6-core CPU system with hyper-threading enabled, they achieved best performance by reducing the thread ratio to 0.125, which corresponded to three threads generated by Suricata. They also determined that the hyper-threading configuration caused variations in the performance results (30% variation between runs) and that it was actually best to configure the multiple threads to run on the same hardware CPU. Then using a tool called perf-tool (code.google.com/p/google-perftools) they determined that, as the number of threads increased, more time was spent waiting for an available lock. Their conclusion was that configuring Suricata to run in RunModeFilePcapAutoFP results in a steady performance increase, whereas RunModeFilePcapAuto shows an initial increase, then a continued decrease in performance as measured by packets per second processed.

Of the previous work, [7] was the most ambitious. However, they used a relatively simple architecture with only server traffic, only six known exploits, and only four processors. It is unclear to what extent their results generalize to more realistic and higher-performance settings. So our goal was to test this, as well as do more careful tests along the lines of [8] and [9]. More details of our experiments are given in [10].

III. Description of Experiments

Thorough testing of a multi-threading intrusion-detection system against a control (in this case, Snort) requires (a) comparison tests of speed in an environment similar to the ones in which it is commonly installed; (b) tests for speed improvements due to multi-threading to justify using it; and (c) tests of the accuracy in detecting known exploits to confirm its coverage. So we conducted three experiments. If Suricata performs at least as well as Snort on each, we can recommend it. The same three tests can be used to evaluate any multi-threading product.

A. Experimental Setup

Network traffic for our experiments came from a tap on the full set of traffic of the U.S. Naval Postgraduate School network, ERN. This had a bandwidth of 20 Gbps, and traffic averages 200Mbps per day. Signatures came from the two sources, the SourceFire Vulnerability Research Team (VRT) rules ("the official rule set for Snort" (SourceFire, 2011)) and the Emerging Threats (ET) rules ("platform-agnostic" additional capabilities (www.emgergingthreats.net)). Both Suricata and Snort support the VRT and ET rules.

Most experiments were conducted in a virtual machine running VMware ESXi 4.1. The server hardware was a Dell Poweredge R710 dual quad-core server with 96 GB of RAM. Each CPU was an Intel Xenon E5630 running at 2.4 Ghz. The data storage was accomplished through three fiber-channel attached RAID 5 configured arrays, supporting the relatively large network traffic capture files (PCAP files) needed to test the detection engines. The server had eight 1Gbps network cards installed, with four reserved for the various management activities and four available for the Virtual Machine. For our experiments the server was configured with two interface cards, one for system administration and one to capture network traffic. The interface card attached to the network tap was configured in promiscuous mode to allow it to receive all of the network traffic. Our virtual machine used 4 CPU cores and 16GB of RAM. The operating system chosen for the experiment was CentOS 5.6 due to its popularity for enterprise applications and close relationship to Red Hat Enterprise Linux.

Installation of Suricata was straightforward. But we needed to compile Suricata ourselves, installing a number of software dependencies which were not included in the CentOS 5.6 distribution. Among these were the PERL compatible regular expression (PCRE) libraries, packet-capture libraries (Libpcap) to allow the operating system to capture all of the traffic on the network, and YAML libraries (yaml.org). During experiments Suricata released a major upgrade to version 1.1beta2 which we switched to. Although the rules we used were intended to work for both Snort and Suricata, we found some exceptions. We received a number of error messages upon loading the VRT rules in Suricata.

Snort 2.9.0.5 has an issue of compatibility with the version of Libpcap that is distributed with CentOS prior to 6.6. Fortunately Vincent

Cojot maintains a series of RPMs (precompiled software for installation on Red Hat-based Linux distributions) for Snort on CentOS (vscojot.free.fr/dist/snort) that include Libpcap. We would have liked to upgrade to the Snort 2.9.1 beta because of the large size of our PCAP files, but Cojot's repository did not support it.

The upper limit on the memory for applications in a 32-bit operating system meant we could not load the entire combined ET and VRT rule set of 30,000 rules in our 32-bit CentOS 5.6 operating system. So our experiments reduced the number of rules to a combination of ET and VRT rules totaling 16,996 signatures.

Pytbull was also used, a utility to test and evaluate an intrusion-detection system's ability to detect malicious traffic (pytbull.sourceforge.net) by sending it sample traffic. To capture live traffic from the network for later replay, Tcpdump (tcpdump.org) and Tcpreplay (tcpreplay.synfin.net) were used. Collection of the system performance data during experiments was done with Collectl (Collectl.sourceforge.net).

B. *The three experiments*

Our first experiment examined the real-time performance of Snort and Suricata while monitoring live backbone traffic from the ERN. Performance data from the CPU, RAM, and network interface was recorded, examined, and compared. The first experiment initially compared Suricata to Snort when monitoring network traffic just inside the border router (Figure 1). We also then ran instances of both Snort and Suricata on the virtual machine at the same time to compare the accuracy of each detection engine on the same live network traffic.

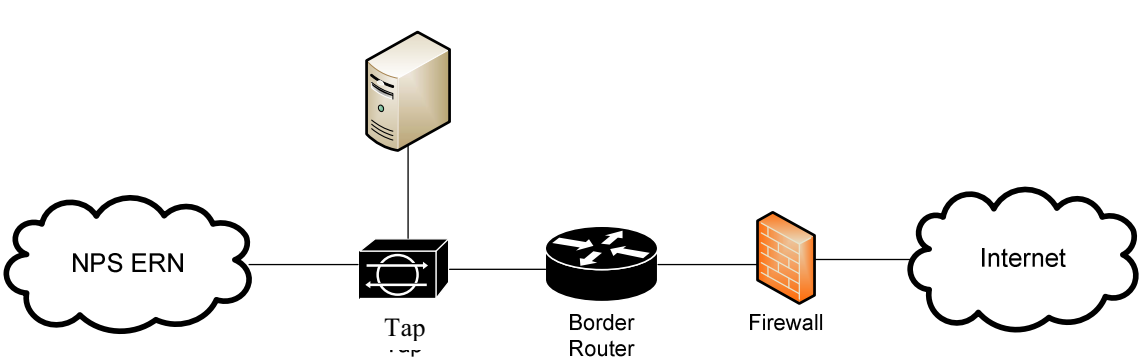

Figure 1. Setup of the first experiment.

The second experiment ran four hours of stored traffic through Suricata on a supercomputer with 48 CPUs. This type of task is important for our Information Technology department as they regularly must do retrospective analysis of attacks. Performance configuration settings for each detection engine were set to the default parameters. The second experiment first used Tcpdump to obtain a large file of PCAP traffic data from the NPS ERN backbone. The file was roughly 6GB and consisted of full packet data for about four hours of traffic. We ran Suricata with this large PCAP file on our Hamming supercomputer with 48 CPUs to compare its performance with that of Experiment 1 with various configuration settings.

The third experiment tested how accurately Suricata and Snort recognized known malicious or irregular traffic. Using Pytbull we sent 54 suspicious or malicious packets through the intrusion-detection systems to stimulate alerts. Nine kinds of tests were conducted: client-side attacks, common rule testing, malformed traffic, packet fragmentation, failed authentication, intrusion-detection system evasion, shell code, denial of service, and malware identification. This experiment required a machine to generate the test traffic and provide HTTP and FTP servers, and a victim machine. We chose Vsftpd for our FTP client, and used the Web server already in CentOS 5.6. The Web server was preloaded with a variety of corrupt files, malware from security-related sites on the Internet that collect them for research purposes. We selected four PDF and one XLS file.

IV. RESULTS

A. *Experiment One*

The data collected showed that Suricata consumed more computational resources than Snort on the same network traffic. When Snort and Suricata were run at different times, CPU use for Snort was 60-70% percent for one CPU while Suricata was 50-60% on each of four CPUs. Suricata was also more memory-intensive than

Snort, and the system memory it required increased considerably over the experiment (Figure 2). Snort's memory usage was quite constant at around 0.9 gigabytes.

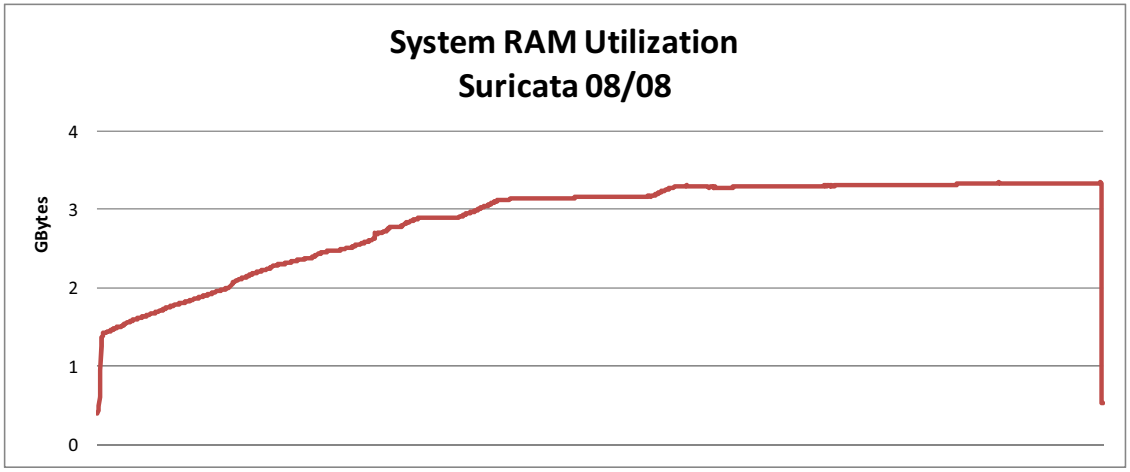

Figure 2. Suricata main-memory usage.

Despite our multicore processors, the systems had difficulties keeping up with network traffic. Using Tcpdump, we first saw dropped packets at rates of up to 50% for both Suricata and Snort in their log files. Given the similar rate in both engines, we concluded that the problem was in the virtual environment of the ESXi server and that the default kernel buffer settings were too small. So we set net.core.netdev_max_backog to 10000, rmem_default to 16777216, remex_mx to 33554432, tcp_mem to '194688 259584 389376', tcp_rmem to '1048576 4194304 33554432', and tcp_no_metrics_save to 1. This reduced the packets dropped by Tcpdump to less than 1%.

However even with these settings, Snort reported a packet drop rate of 53%. Suricata, on the other hand, had a drop rate of 7%. The Snort log file further classifies the dropped packets as "outstanding" meaning dropped before being received by the packet processing engine [11]. Our data showed that the number of outstanding packets in Snort matched the number of dropped packets in Snort, indicating that the loss of packets occurred prior to packet capture, and is therefore not a function of the processing load of the detection engine itself. Suricata does not break down the composition of dropped packets in the same manner as Snort, so the same deduction cannot be assumed solely by the log files. Further investigation should be conducted to determine why there is a disparity in drop rates between Tcpdump, Suricata, and Snort.

A second part of the first experiment compared the alerts generated by Suricata and Snort running simultaneously and looking at the same traffic. For most rules Suricata generated more alerts (0% to 30% more) than Snort on the same network traffic. Though both engines loaded the same rule sets, we did get some error messages in loading and some rules may have failed to load successfully. Other reasons could be differences in the implementation of rules.

B. Experiment Two

The second experiment ran Suricata on a high-performance computer, a Sun 6048 system with 144 blades and 1152 CPU cores. We used one compute node composed of 48 AMD Opteron 6174 12-core processors with 125GB of RAM available. The operating system was CentOS 5.4. Installation of Snort and Suricata was straightforward. For these experiments we used a 6GB Libpcap file previously generated from NPS backbone traffic with an average packet size of 803.6 bytes. We did not study the accuracy of the alerts for this experiment, only the relative difference in processing speed.

We adjusted three parameters in the Suricata configuration file to tune the performance: detect_thread_ratio, max-pending-packets, and run mode. The detect_thread_ratio value determines the number of threads that Suricata will generate; it is multiplied by the number of CPUs to determine the number of threads, and we used values from 0.1 to 2.0. The max-pending-packets value determines the maximum number of packets the detection engine will process simultaneously; there is a tradeoff between caching and CPU performance as this number is increased. We used values of 50, 500, 5000, and 50,000 in our experiments. The runmode value determines how Suricata will handle the

processing of each thread. The options are single (single-threaded), auto (multi-threaded), and AutoFP (multi-threaded but not splitting flows).

Results showed that with 48 CPUs, the performance in Auto runmode varied slowly from 28K to 32K packets per second as max-pending-packets increased from 50 to 50,000, whereas in AutoFP runmode it increased much more from 15K to 135K. The minimum packets per second processed by Suricata corresponds to 108 Mbps and the maximum to 854 Mbps. But using 4 CPUs, Auto ran 19K-22K and AutoFP 17K-18K. The number of threads, set by the detect_thread_ratio parameter, had little impact on performance in either mode with 48 CPUs. But it did for 4 CPUs (Figure 3). The noticeable drop in performance at 8 threads while in the AutoFP runmode (the darker lines) is likely the result of limited system memory causing the operating system to begin using the hard drive swap space to augment the main memory.

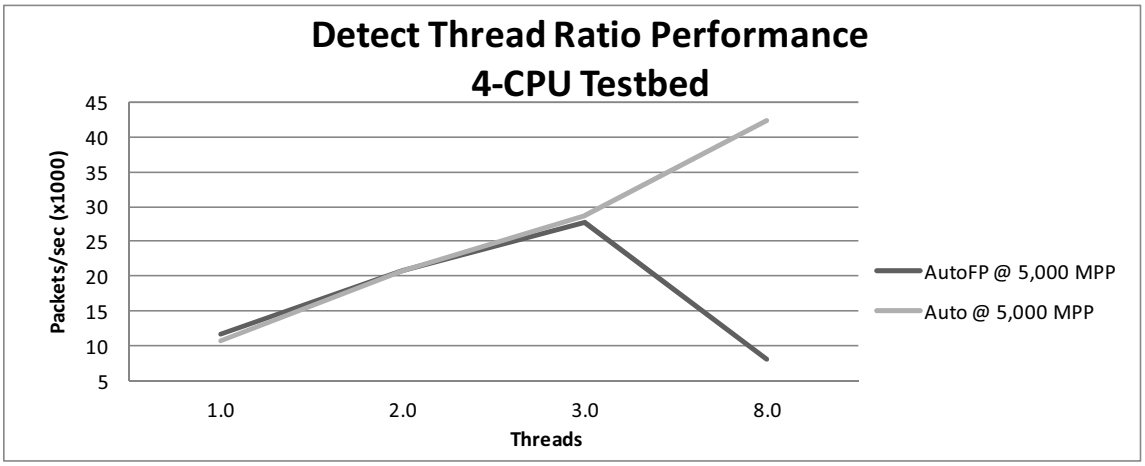

Figure 3. Suricata packet processing rate with number of threads, on the high-performance computer with four CPUs.

C. *Experiment Three*

The third experiment sent known malicious packets to Suricata and Snort installations in 54 tests in 9 categories. There were only a few clear false negatives. One was a client-side attack where Suricata detected all 5 tests and Snort only 3, and the other was in the evasion-technique attack where Snort identified that an evasion attempt was underway while Suricata did not. However, recognizing false positives was not easy because in many cases an alert was generated that was not closely related to the actual attack. For example, Test 8 under the category Test Rules is a full SYN scan; a true positive would be an alert that a SYN scan was underway. However, we got a large number of more specific alerts instead for the steps of the scan. These alerts would still be useful in monitoring, so we labeled these as "grey positives". But there were still clear false positives, such as an alert for a Trojan infection during an SYN scan.

Overall, Snort had 16 false negatives, 10 false positives, 1720 "gray positives" (including 1640 "evasion techniques"), and 44 true positives. Suricata had 12 false negatives, 8 false positives, 1449 gray positives (including 1275 evasion techniques), and 81 true positives. The evasion techniques involved five port scans, so the amount of traffic generated for this test was significantly more than that of any other test.

The recall fraction was 0.73 for Snort and 0.87 for Suricata in our third experiment. We can calculate two kinds of precision, one excluding the grey positives as false positives and one including them. We got 0.81 for Snort and 0.91 for Suricata excluding them, and 0.036 and 0.052 including them. More details are in [10].

V. CONCLUSIONS

Overall, Suricata did at least as well as Snort in our tests, and sometimes better, contradicting some of the results of [7] in less elaborate tests. Suricata's false positives and false negatives can be attributed to weaknesses of the rule set used for the tests. It was inconclusive from our tests whether Suricata or Snort has a better detection algorithm. But a 64-bit machine is recommended for both to enable loading the full rule sets.

The first experiment showed that the aggregate CPU use of Suricata was nearly double that of Snort, and Suricata used over double the amount of RAM used by Snort, apparently to handle the multi-threading capability. But Suricata can take advantage of the extra threads to reduce the rate of dropped packets. Suricata also can grow to

accommodate increased network traffic without requiring multiple copies of the product. Snort is lightweight and fast but limited in its ability to scale beyond 200-300 Mbps network bandwidth per instance. While Snort's processing overhead is less than that of Suricata, the need for multiple instances to accomplish what Suricata can achieve with its multi-threaded design elevates the cost to operate and manage a Snort environment.

The first experiment also showed the dangers of using virtualization to conduct experiments on an intrusion-detection system that is operating on significant traffic levels, since Snort could not keep up with our traffic once virtualized though it does fine without it. Virtualization made it easier for us to conduct experiments but is not necessary for routine monitoring.

The second experiment showed that Suricata could obtain a big improvement in its performance on 48 CPUs when high-performance computers are available. But care is needed in setting the parameters before this improvement can be seen. The third experiment showed high rates of detection of known malicious packets by both engines. However, Suricata made a few errors that Snort did not. The blame must lie in either our inability to load a few of the Snort rules in Suricata or else in bugs in Suricata's implementation.

A key issue with Suricata is that it is still being developed. During our research there were three minor version changes (1.0.3, 1.0.4 and 1.0.5) and two beta versions (1.1 beta1 and 2), and each version contained significant improvements to the previous version. In comparison, Snort has been on the same production release (2.9.0.5) for five months. Software requiring frequent upgrades is not an optimal choice for a production environment, so that is a weakness of Suricata. Nonetheless, the pace of upgrades is likely to slow and Suricata should be more reliable.

A key goal of this research was to make an informed recommendation to our Information Technology department on the implementation of Suricata. The ability to use multi-threaded techniques in a multiple-CPU environment will give Suricata an advantage over Snort in the future as networking needs in our organization continue to increase. But Snort is still very capable and should remain in use within our production environment for the immediate future until Suricata becomes more stable. Of course, network intrusion-detection systems are just one security technology, and we must also incorporate host-based and anomaly-based intrusion detection, and access controls at many levels, to defend our networks.

The methodology from our research should be useful in comparing other intrusion-detection products. Testing for a few hours on the entire traffic of our organization gave us a good view of the challenges that need to be faced by the software. It was clearly useful to do both tests on real traffic and tests on known malicious packets. It was also useful to separate analysis of real traffic from tests of multiprocessing capabilities, best done with retrospective analysis to eliminate inconsistent random fluctuations in traffic.

ACKNOWLEDGMENT

Views expressed in this paper are those of the authors and do not represent those of the U.S. government. Thanks to Simon McClaren and Chris Gaucher for help in setting up the NPS tap.

REFERENCES

[1] D. Kuipers and M. Fabro, "Control System Cyber Security: Defense in Depth Strategies," No. INL/EXT-06-11478, 2006

[2] G. Kumar and S. Panda, "Spectrum of Effective Security Trust Architecture to Manage Interception of Packet Transmission in Value Added Network," The Global Journal of Computer Science and Technology, Vol. 11, No. 4, pp. 73-77, 2011.

[3] B. Caswell, J. Beale, and A. Baker, Snort IDS and IPS Toolkit. New York: Syngress, 2007.

[4] M. Lococo, "Capacity Planning for Snort," retrieved October 2, 2011 from mikelococo.com/2011/08/snort-capacity-planning/.

[5] T. Weber, "Network Iintrusion Detection - Keeping Up with Increasing Information Volume," SANS Institute, 2001.

[6] P. Mell, V. Hu, R. Lippmann, J. Haines, and M. Zissman, "An Overview of Issues in Testing Intrusion Detection Systems.," retrieved October 4, 2011 from www.net-security.org.

[7] D. Day and B. Burns, "A Performance Analysis of Snort and Suricata Network Intrusion Detection and Prevention Engines," Fifth International Conference on Digital Society, Gosier, Guadeloupe, pp. 187-192, 2011.

[8] S. Damaye, "Suricata-Vs-Snort," retrieved from www.aldeid.com/wiki/Suricata-vs-snort, October 2, 2011.

[9] E. Leblond, "Optimizing Linux on Multicore CPUs," retrieved September 18, 2011 from home.regit.org/2011/01/optimizing-suricata-on-a-multicore-cpu/.

[10] E. Albin, "A Comparative Analysis of the Snort and Suricata Intrusion-Detection Systems," M.S. thesis, U.S. Naval Postgraduate School, September 2011.

[11] M. Watchinski, "Unusual Snort Performance Stats," retrieved October 2, 2011 from comments.gmane.org/gmane.comp.security .ids.snort.general/30527.