These go to eleven: Cranking up the knobs on IDS scaling performance

Sunny James Fugate
University of New Mexico
Department of Computer Science
Albuquerque, New Mexico

ABSTRACT

Signature-based intrusion detection system (IDS) approaches represent the brunt of modern threat detection methods. This is primarily due to their specificity and low false-positive rates and in spite of scalability issues. The inherent scaling issues have meant that measurements of these systems generally ignore the scaling of systems beyond conventional parameter spaces. In particular, while signature-based systems are conventionally measured for total packet processing throughput and false alarm rates, performance is highly dependent on ruleset size. While IDS packet processing performance may appear to be well understood, IDS scaling performance has not been adequately characterized beyond available rulesets. In this paper I present my measurement methods, describe a straightforward method for generating large random rulesets, and present an analysis of the scaling performance of the Snort IDS system.

1. INTRODUCTION

Signature-based detection systems rely on explicit patterns within input traffic. Such approaches offer very precise detection of known threats at the cost of poor recall and poor coverage in the face of new threats and new variants of old threats. On many networks, *qualified events* (those resulting in an alert within the IDS) account for only a small portion of total network traffic. As a result, the usable coverage of signature-based IDS systems is severely limited. Expanded coverage and partial match information might be retained by an IDS if it were not for poor performance scaling.

Commonly, commercial systems must often be tuned for a particular environment to achieve acceptable performance[1]. The most common "tuning“ method is simply to remove unused signatures. The costs of performing a signature-based detection are roughly proportional to the sum of packet length and total signature length[2]. As network bandwidths have increased, so have the number and variants of threats, driving the need for expanded coverage, but remaining bounded by proportional increases in detection costs. It is the belief of the author that the high cost of modern IDS, is inextricably due to poor coverage. If performance scaling of these systems is well-understood, methods may be identified to ameliorate long-term performance degradation.

The contribution of this paper is an experimental analysis of Snort IDS scaling performance as ruleset size is varied. In addition to performance scaling, the frequency distribution of events is assessed and an assessment of IDS alerts in respect to their information content. The remainder of this paper reviews prior research in the field, defines a set of metrics and cost functions for describing performance scaling of signature-based systems, and describes the result of series of experiments used to determine the scaling performance of recent versions of the widely used Snort IDS.

2. BACKGROUND

2.1 Signatures & Coverage

Signature-based IDS are precise and due to their simplicity have been employed within many large-scale, commercially available systems. In many systems each input (e.g. header, packet, TCP session, event, event sequence, derived feature set, etc.) is compared against sets of thousands of signatures in a more-or-less brute-force manner. The patterns used and the algorithms employed for pattern matching are very efficient. Significant gains have been made in the last two decades in the area of pattern matching, leading to substantial performance improvements[3, 4, 5, 6, 7].

Commonly, signatures are written to be very precise with respect to vulnerabilities, exploits, and other “known bad" sequences. False negatives are common and false-positives are often manually “tuned" out of rulesets over time[11]. Such systems have the benefit of precision, but are generally poor at detecting new exploits and are very labor intensive to maintain in the face of large numbers of vulnerabilities and exploits. The rule-tuning process often consumes a significant portion of the time spent managing a signature-based system’s ruleset. Adaptive systems have been proposed and constructed to alleviate some of these issues[12, 13, 14]. In many instances adaptation occurs only in direct response to operator feedback in identifying false-positives or operator information overload.

There has been a number of independent studies regarding the expansion of IDS coverage. Aikelin et al. describe an approach to expanding Snort’s coverage of previously undetected variants by relaxing and varying signature parameters[10]. In their approach, rules are generalized by allowing lower-priority partial matches when most of the features of a rule are matched. By generalizing each rule in this way, expanded coverage is gained. Other approaches by Brumley et al. have focused on automated generation of vulnerability signatures[15, 16]. This has the benefit of potentially catching zero-day attacks, but appears to be limited to host-based detection systems. Network-based detection has also benefited by the advent of automated signature generation, such as the polymorphic worm signature approach by Zhang et al..[17] or the use of honeypots and attack fingerprints by Portokalidis et al[18]. As automated signature generation techniques evolve, the need for improved IDS detection performance characteristics become paramount.

2.2 Costs & Concessions

When implementing an IDS some form of cost analysis (e.g. computing, latency, hardware, training, etc.) is generally performed in order to choose the right IDS technology and ruleset for a given network. There are many trade-offs which result in sub-optimal de-

tection, but which decrease the IDS cost substantially. Many modern IDS also rely on labor-intensive tuning and rule-refinement to match particular network characteristics and known host vulnerabilities. While improving performance, this process also introduces a greater chance of false negatives. Work by Fan, et al. describe cost metrics in terms of operational costs, attacker induced damage, and incident response, citing the need to consider cost within the development and deployment of IDS[13]. Others have cleverly incorporated cost assessment into decision support systems to propose or enact response actions[19], IDS reconfiguration, and dynamic performance tuning[14].

Cost-driven optimizations allow IDS to be tractable given limited computing resources. However, they can easily be based on tenuous assumptions. Any risk analysis which performs this type of trade-off analysis needs to consider the possibility of attackers gaming a system using knowledge of financial or computational concessions. For example, the rule ordering of Snort and other signature-based IDS are partially under user control and partially under the control of engineering optimizations. This ordering matters. For example, the Snort IDS, by default only returns the first (or at best a limited number) of alerts for a packet in order to eliminate as many comparisons as possible [20]. Careful management of ruleset in order to achieve desired performance goals may also mask performance issues and can actually decrease the usefulness of IDS by removing contextual knowns from the stream of true-positives being displayed to an analyst. In a perfectly "tuned" system one might expect only the most high-priority events to be displayed and all other events to be discarded (or at least hidden). If it were at all possible to perform intrusion detection without making such concessions, we would greatly simplify the task of the IDS designer (or maintainer). The challenge is achieving these goals while also improving capabilities and performance.

2.3 Performance Measurement

IDS experimentation and test has tended to focus on three characteristics, namely: detection performance, resource usage, and resilience[21]. Detection performance uses ground truth data to determine detection rates: the true and false positives and negatives detected by a system. The resources used by a particular system can be measured as the average computing cost per packet or stream processed, where packet sizes are specified as part of the test criteria. Resilience has a broad array of meanings: performance in resource constrained environments; effects of resource contention; effects of high alarm rates during abnormal attacks; losses incurred during high network loads; resilience to artificial attacks used to mask attacker activities; and resilience to various attacks to the IDS itself[21].

It is important to note that IDS performance characteristics are generally not independent (though many experimenters have treated them as such). Of primary importance is the coupling of resource utilization and detector performance, performance degrading as contention for the CPU or other shared resources increases[22, 23]. Similar to the approach given in this paper many studies also consider the affect of ruleset size on successful packet processing rates for a given system[22].

The use of synthetic or generated attack data is also common for assessing IDS performance and resilience[24, 25]. Privacy and confidentiality concerns are the primary motivation. As such, creating and making available labeled datasets is no easy task. Synthetic approaches are not without their pitfalls, occasionally leading to erroneous conclusions. Other testing approaches (such as those used within the DARPA 1998 Evaluation) use real traffic on real networks, but may still mis-characterize IDS performance due to closed network topologies and unrealistic attack sequences[26]

I have taken a different perspective on IDS performance and have chosen to measure its scaling performance in respect to ruleset sizes well beyond rulesets which are commonly available. Within the following experiments, the size of the IDS ruleset is intended to serve as a proxy for IDS coverage. The actual coverage of an IDS is is difficult to measure given that changing environments, vulnerabilities, and network traffic each may contribute to whether alerts occur or are relevant. I am concerned with the long-term scalability of such systems as new threats emerge and new rules are added.

3. IDS SCALING PERFORMANCE

3.1 Generating Large Rule-sets

An obvious issue with testing an IDS's rulesets size scaling performance is that there don't exist that many rules in the wild. A significant deviation from traditional testing was the introduction of a randomly generated ruleset. This was necessary in order to test the performance of the system beyond the scale and scope of available signature sets. It was desirable that the generated rules were statistically similar to the existing ruleset in respect to the string and regular expression features used. For the purposes of generating a large number of random rules, a straightforward algorithm using non-parametric statistics was used to generate a ruleset of approximately 800K rules. It is important to note that the random rules do not have any semantics and are solely used to test scalability of the IDS. As such, traditional performance measures (such as rates of true positives and false negatives) are meaningless.

To generate a large number of random rules: First, an existing ruleset is split into individual features and each feature appended to a file according to its label. Each labeled file contains as many duplicates and unique features as there are duplicates and unique features in the actual ruleset, totaling approximately 27,000 unique features. Second, based on the number of rules desired, each rule in an actual Snort ruleset is permuted thousands of times by replacing each labeled feature with a sample drawn randomly from the labeled file for the feature. This was done in order to retain a similar distribution of the feature labels and total number of features that were used. As a result, random rules were generated that have the same distribution of features and feature values as rulesets that are normally used. Lastly, invalid and duplicate rules were removed by eliminating rules which did not pass Snort's rule parser.

One issue with this approach is that many invalid and duplicate rules are generated and must be removed in order for the ruleset to be used. Of 1.2M rules randomly generated, only 800K could be retained after removal of duplicates. This leads to the random rules being slightly biased towards more complex rules with a larger number of features. As this results in an overestimate of performance costs it is not an issue in respect to my research goals.

3.2 Measuring Performance Costs

Although the true cost function for performing detection using a given IDS configuration and computing system is unlikely to be known, it can be estimated using experimental measurements. In particular, we can easily define a performance metric incorporating cost functions C_{time} (total CPU time) and C_{loss} (percent packet loss) and parameterized by the signature set size $|n|$. The cost functions can be multiplicatively combined to mean CPU-time per % of packets processed, appropriately penalizing high CPU-time or high packet loss. Note that the cost function will change (particularly with respect to packet drop rates) as the line speed is decreased.

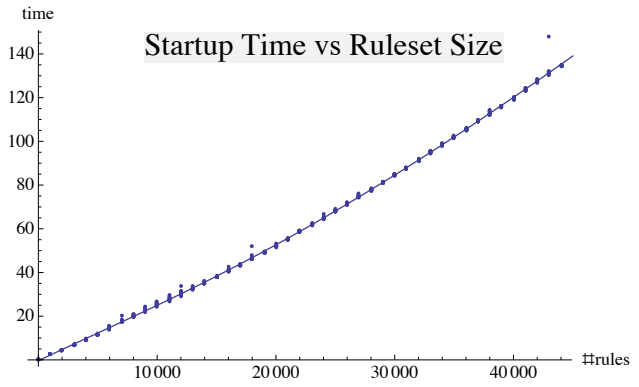

Figure 1: 5th order polynomial fit for Snort's startup time as measured in seconds. R^2: 0.9998.

$$C(n) = C_{time}(n) \cdot C_{loss}(n) = \frac{C_{time}(n)}{1 - C_{loss}(n)} \quad (1)$$

The cost functions C_{time} and C_{loss}, need to be experimentally determined. Depending on the application, the cost functions should represent conservative estimates of the actual performance costs.

My test system was an Ubuntu Linux 11.04 system running on a pair of 2GHz Dual-core AMD Operation processors, 28GB of RAM, and consumer-grade Broadcom BCM5780 Gigabit ethernet cards. The default Snort configuration (for version 2.9.0.3) was used, but with secondary detection engines disabled or suppressed. When a traditional ruleset was used, it was using the SourceFire® provided "VRT" release downloaded on Oct. 17, 2011.

My measurement approach obtains functional forms of the cost functions for a particular system and IDS configuration. Many of the performance characteristics of IDS systems are highly dependent on system and software configuration[22]. Even small changes in configuration or run-time options can have significant effects on overall performance[23]. Hardware configuration, system settings, and software configurations were kept consistent between tests.

3.2.1 Startup Performance

One of the first issues discovered in the current version of Snort is extremely slow startup times for large ruleset sizes. Startup performance is relevant as it can result in experiment run-time being dominated by startup time. As a result, experiment sizes were kept relatively small ($< 50K$ rules) in respect to the ruleset available ($\sim 800K$ rules). Startup time was also required so that measurements could be delayed until the IDS was ready for input.

An experiment was run to determine startup time which would limit experiments. This experiment made use of the Linux "time" program while running Snort against a small file containing a small sample of 10 packets. The packet processing time in this case did not have a measurable impact on the results for the sizes of rulesets tested. 10 trials were run at increments of 1,000 rules for each ruleset size between 0 and $45,000$ ($n = 450$). Figure 1 shows a best-fit polynomial function for Snort's startup time on my test system. A single test run using 100K rules required approximately 17 minutes which corresponded closely to the polynomial fit. Consequently, the current startup performance hinders experimentation with extremely large rulesets. Large rulesets might still be used, but would need to be split between multiple Snort instances to achieve reasonable startup times. It is also possible that there are configuration options which may alleviate the issue, though these are not known.

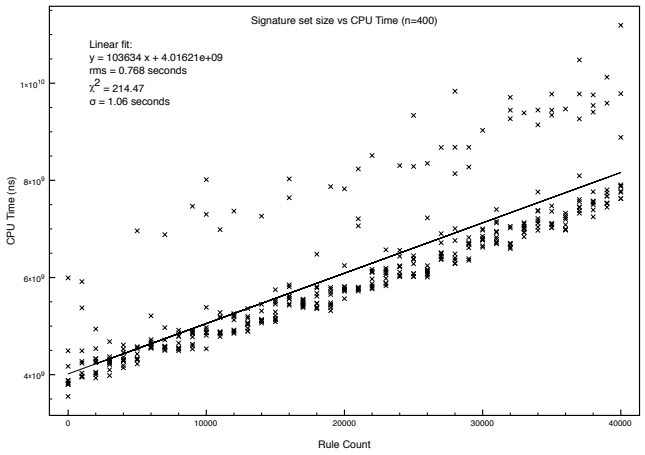

Figure 2: Linear scaling of total CPU time of the Snort IDS with ruleset size. Test system was Linux Ubuntu 11.04, running on a pair of 2GHz Dual-core AMD Opteron processors.

3.2.2 CPU Time Scaling Performance

Since we are comparing the cost of Snort instances running with various sized rulesets, we can gauge the computing cost by measuring the performance on a test platform as ruleset size is increased and finding an acceptable function to model the system's behavior as shown in Figure 2.

A linear fit resulted in a simple estimated cost function in terms of nanoseconds of CPU time:

$$C_{time}(x) = 103634x + 4.01621 \times 10^9 \quad (2)$$

The y-intercept represents the total CPU-time for the entire system when the IDS isn't doing anything. The slope of the function represents the cost in nanoseconds of CPU-time incurred due to ruleset size increases. It was expected that the scaling performance of the Snort IDS was linear in respect to ruleset size as the version of Snort being assessed uses an Aho-Corasick algorithm for string matching[2, 6]. This algorithm's complexity is linear in the sum of the length of patterns, the string being matched, and output length[28]. Snort's resource usage appears to scale linearly as shown in Figure 2 with the glaring exceptions of startup time and packet drops as described in the next section.

3.2.3 Packet Drop-rate Scaling Performance

Unfortunately, estimating performance by simply measuring total system CPU-time is eventually confounded by shared-resource issues when measuring performance of Snort running with large rulesets. Large rulesets, while incurring only a proportional increase in CPU-time, result in substantial packet loss on the front-end of the IDS. This finding is the reason that packet loss is included in the cost function for the system. Without considerations of packet loss we might only slightly overestimate performance costs for small rulesets but we would grossly underestimate performance costs for larger rulesets.

Several experiments were run to learn the packet drop rates as a function of ruleset size. The randomly generated ruleset was used to determine a worst case scaling rate. For these tests, the alerting and logging facilities of Snort were disabled. In this way Snort will not block due to I/O constraints on output alerting and logging costs. This is particularly relevant when measuring performance

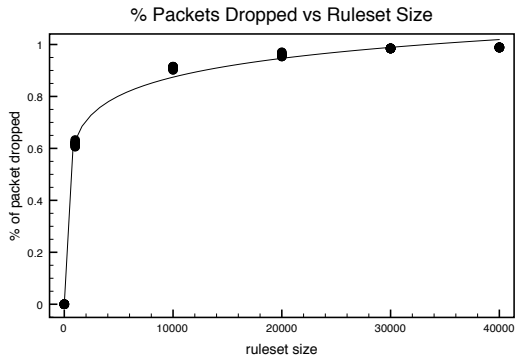

Figure 3: Packets dropped by Snort as ruleset size increases (n=60). Packets were replayed at the network card's maximum throughput ($\sim$ 400 Mbps).

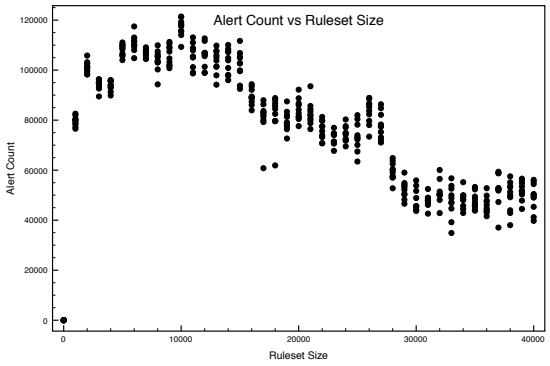

Figure 4: 10 trials at each of 1K through 40K ruleset sizes for a 1M packet sample on a live ethernet interface. Packets were replayed at the maximum interface speed ($\sim$ 400 Mbps). The non-monotonicity is due to high packet drop rates. Coverage measurements must therefore be based on cached PCAP data to eliminate drops from I/O blocking and CPU contention.

in respect to ruleset size as even small sets of randomly generated rules are likely to produce a larger number of alerts than conventional rules.

Figure 3 demonstrates that while rulesets of size n increase the system overhead proportionally, the larger complexity of the ruleset increases packet drop rates by up to a factor of 10. The cause for the high packet drop rate is likely that the Snort process is starved for CPU-time due to processor contention or conflicting I/O IRQs, possibly due to configuration issues such as a high NAPI (New API) budge rate as suggested in [23]. The unfortunate side effect is that the total number and type of alerts produced is not monotonically increasing as ruleset size increases (see Figure 4).

High packet-loss issues have been noted by others and various methods have been used to limit packet loss[23]. In our case, however, as the number of rules is increased, contention for the CPU will eventually result in the same problem. As a result, the overall shape of the packet loss function is unlikely to change significantly, though this has not been thoroughly explored.

If we measure Snort's packet processing rate using the built-in performance monitor we can estimate the performance bottleneck as an exponential function in terms of the percentage of received packets processed. On my test system, an acceptable functional

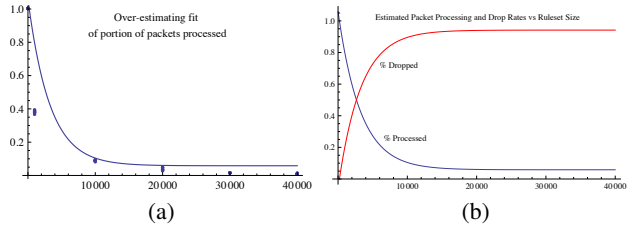

Figure 5: (a) Over-estimating fit for packet processing rate. (b) Curves estimating packet processing rate as ruleset size is increased.

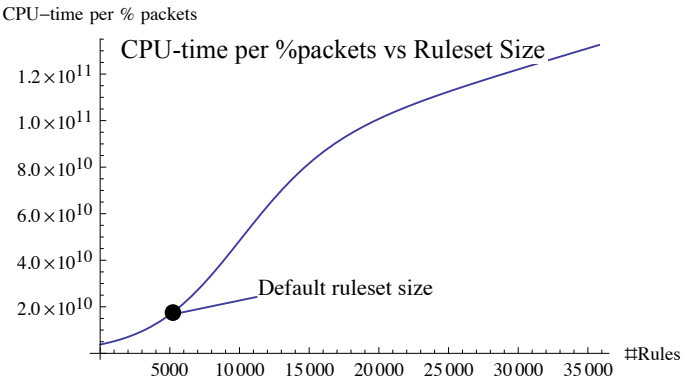

Figure 6: The cost function for the Snort IDS running on a test system. CPU Time per percent packet units are nanoseconds per %.

model was found:

$$PacketProcessingRate \leq 1.0392^{-0.008x} + 0.0583 \quad (3)$$

As can be seen from Figs. 5a and 5b, this fit is a gross over-estimation of experimental measurements, but sufficient for our purposes. An over-estimation of the number of packets successfully processed will result in an under-estimate of the effects of increasing the number of rules in a particular IDS instance. For my test system the packet-loss rate is:

$$C_{loss}(n) \geq 0.9417 - 1.0392^{-0.008n} \quad (4)$$

It is important to reiterate that other architectures, operating system kernels, and configurations will result in different scaling performance, though the basic functional shape is likely to remain similar. Figure 6 shows the scaling performance of the Snort IDS for my test system. Note that the default ruleset size is well below the knee of the performance curve.

4. DISCUSSION & FUTURE WORK

The basic measurement approaches presented here are not novel except for the explicit representation of scaling performance beyond traditional ruleset sizes. As is the case with many scaling performance studies, I have simply abused the IDS system by forcing it to perform detection outside of its design parameters. However, the end goal of all of this hand-wringing over IDS performance scaling is to show how the performance characteristics are amenable to new optimization methods[30, 31]. The promise is that much larger

rulesets might use the less computing power by trading an increase in unlikely false negatives for smaller detector costs. For prediction to play a significant role in improving IDS performance, new rulesets with superior coverage are needed along with new performance models and testing methods.

5. REFERENCES

[1] N. Stakhanova, Y. Li, and A. A. Ghorbani. Classification and Discovery of Rule Misconfigurations in Intrusion Detection and Response Devices. Proceedings of the 2009 World Congress on Privacy, Security, Trust and the Management of e-Business. 2009).

[2] V. Dimopoulos, I. Papaefstathiou, and D. Pnevmatikatos. A Memory-Efficient Reconfigurable Aho-Corasick FSM Implementation for Intrusion Detection Systems. Proceedings of the 2007 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation. pp. 186-193. (2007).

[3] S. Kim. Pattern Matching Acceleration for Network Intrusion Detection Systems. SAMOS'05: Proceedings of the 5th international conference on Embedded Computer Systems: architectures, Modeling, and Simulation. (2005).

[4] C.-H. Lin and S.-C. Chang. Efficient pattern matching algorithm for memory architecture. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 19, no. 1, pp. 33-41. (2011).

[5] D. Luchaup, R. Smith, C. Estan, and S. Jha. Multi-byte regular expression matching with speculation. Recent Advances in Intrusion Detection. (2009).

[6] M. Norton. Optimizing Pattern Matching for Intrusion Detection. Technical Report, SourceFire Inc (2004).

[7] N. Schear, D. R. Albrecht, and N. Borisov. High-Speed Matching of Vulnerability Signatures. LNCS, vol. 5230, no. 2008, pp. 155-174. (2008).

[8] L. Vespa, M. Mathew, and N. Weng. Predictive Pattern Matching for Scalable Network Intrusion Detection. Information and Communications Security. LNCS, vol. 5927, pp. 254-267. Springer, Heidelberg (2009).

[9] G. Tripp. A Finite-State-Machine based string matching system for Intrusion Detection on High-Speed Networks. Proceedings of EICAR. (2005).

[10] U. Aickelin, J. Twycross, and T. Hesketh-Roberts. Rule Generalisation in Intrusion Detection Systems using Snort. International Journal of Electronic Security and Digital Forensics. (2008).

[11] I. Dubrawsky and R. Saville. SAFE: IDS Deployment, Tuning, and Logging in Depth Authors. CISCO SAFE Whitepaper, pp. 1-58. (2003).

[12] Z. Yu, J. Tsai, and T. Weigert. An adaptive automatically tuning intrusion detection system. ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 3, no. 3. (2008).

[13] W. Fan, W. Lee, S. Stolfo, and M. Miller. A multiple model cost-sensitive approach for intrusion detection. LNCS, vol. 1810, pp. 142-154. (2000).

[14] W. Lee, J. Cabrera, A. Thomas, N. Balwalli, S. Saluja, and Y. Zhang. Performance adaptation in real-time intrusion detection systems. In Proceedings of the 5th International conference on Recent advances in intrusion detection, vol. 2416, pp. 252-273. (2002).

[15] D. Brumley, J. Newsome, D. Song, Hao Wang, and S. Jha. Towards Automatic Generation of Vulnerability-Based Signatures. 2006 IEEE Symposium on Security and Privacy, pp. 2-16. (2006).

[16] D. Brumley, J. Newsome, and D. Song. Theory and techniques for automatic generation of vulnerability-based signatures. IEEE Transactions on Dependable and Secure Computing, vol. 5, no. 4. (2008).

[17] J. Zhang, H. Duan, L. Wang, Y. Guan, and J. Wu. 2008 International Conference on Computer and Electrical Engineering. in 2008 International Conference on Computer and Electrical Engineering (ICCEE), 2008, pp. 8-13. (2008).

[18] G. Portokalidis, A. Slowinska, H. Bos, G. Portokalidis, A. Slowinska, and H. Bos. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. ACM SIGOPS Operating Systems Review, vol. 40, no. 4, pp. 15-27. (2006).

[19] C. Strasburg, N. Stakhanova, S. Basu, and J. Wong. Intrusion response cost assessment methodology. Proceedings of the 4th International Symposium on Information, Computer, and Communications Security. (2009).

[20] M. Roesch and C. Green. SNORT Users Manual 2.8.6. pp. 1-191. (2010).

[21] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. A. Olsson. A methodology for testing intrusion detection systems. IEEE Transactions on Software Engineering, vol. 22, no. 10, pp. 719-729. (1996).

[22] L. Schaelicke, T. Slabach, B. Moore, and C. Freeland. LNCS, vol. 2820, no. 9. pp. 155-172. Springer, Heidelberg (2003).

[23] K. Salah and A. Kahtani. Improving Snort Performance Under Linux. Communications, IET, vol. 3, no. 12, pp. 1883-1895. (2009).

[24] J. Haines, R. Lippmann, D. Fried, and M. Zissman. 1999 DARPA intrusion detection evaluation: Design and procedures. Technical Report ESC-TR-99-061, MIT (2001).

[25] D. Mutz, G. Vigna, and R. Kemmerer. An Experience Developing an IDS Stimulator for the Black-Box Testing of Network Intrusion Detection Systems. Proceedings of the 19th Annual Computer Security Applications Conference. pp. 374-383. (2003).

[26] M. V. Mahoney and P. K. Chan. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. Recent Advances in Intrusion Detection. (2003).

[27] J. Novak and S. Sturges. Target-Based TCP Stream Reassembly. Technical Report, SourceFire Inc (2007).

[28] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. Communications of the ACM, vol. 18, no. 6, pp. 333-340. (1975).

[29] S. Sen. Performance Characterization & Improvement of Snort as an IDS. Technical Report, Princeton University (2007).

[30] S. Fugate. Using Prediction to Improve the Performance of Network Intrusion Detection. Proceedings of the 2011 UNM Computer Science Student Conference. pp. 65-69. (2011).

[31] S. Fugate. Go for broke: Speculatively Bootstrapping Better IDS Performance. Proceedings of Research in Attacks, Intrusion, and Defenses. (2012). –Submitted