The use of Flow Features in Lossy Network Traffic Compression for Network Intrusion Detection Applications

Sidney SMITH
Computational Information Sciences Directorate, U.S. Army Research Laboratory
Aberdeen Proving Ground, MD 21005, U.S.A

and

Robert J. HAMMELL II
Department of Computer and Information Sciences, Towson University
Towson, MD 21252, U.S.A

ABSTRACT

In distributed network intrusion detection applications, it is necessary to transmit data from the remote sensors to the central analysis systems (CAS). Transmitting all the data captured by the sensor would place an unacceptable demand on the bandwidth available to the site. Most applications address this problem by sending only alerts or summaries; however, these alone do not always provide the analyst with enough information to truly understand what is happening on the network. Lossless compression techniques alone are not sufficient to address the bandwidth demand; therefore, some form of lossy compression must be employed. Working on the theory that a network flow that is malicious will manifest this maliciousness early, we explore the impact of compressing network traffic by stopping the transmission of packets in a flow once a given threshold either in number of packets or number of bytes have been transmitted.

Keywords: compression, network intrusion detection, flow, packet count, byte count

1. INTRODUCTION

Distributed Network Intrusion Detection Systems (NIDS) allow a relatively small number of analysts to monitor a much larger number of sites; however, NIDS require information to be transmitted from the remote sensor to the central analysis system (CAS) [1] as pictured in Fig. 1. This transmission typically uses the same channels that the site uses to conduct business. It is important to reduce the amount of information transmitted back to the CAS to minimize the impact that the NIDS has on daily operations as much as practical.

Smith and Hammell [1] proposed creating a lossy compression tool using anomaly detection techniques to rate each session and a modification of the Kelly criterion [2] to select how much traffic from each session to return as seen in Fig. 2.

The contribution of this research is to explore one method to compress network traffic without unacceptably impacting the ability of the NIDS to detect and analyze malicious activity. It considers the hypothesis that malicious network flows will manifest their maliciousness early. This research examines the implications of only transmitting the packets in a flow up to a threshold in either packets or bytes by comparing the results of a popular network intrusion detection tool before and after compression.

The remainder of this paper is organized into the following sections: Section 2 provides background, Section 3 outlines the approach chosen to address this problem, Section 4 presents our results, and Section 5 provides a conclusion and discussion of future work.

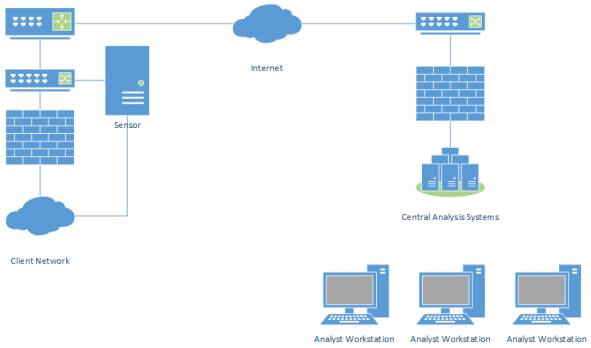

Figure 1. Distributed network intrusion detection

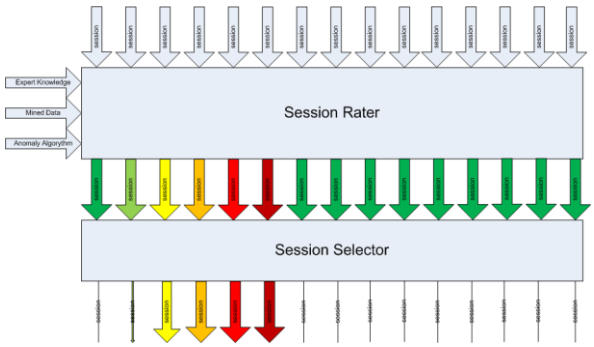

Figure 2. Kelly compressor

2. BACKGROUND

One popular strategy for implementing a distributed NIDS is to do all of the intrusion detection on the sensor and send only alerts or logs to the CAS. [3] [4] A second strategy might be to use lossless compression to reduce the size of the data returned

to the CAS. A third strategy is to implement some form of lossy compression algorithm to send back relevant portions of traffic.

There are three problems with the first strategy. The first is that it has the potential to overburden the sensor's central processing unit (CPU) and introduce packet loss. Smith *et al.* discovered that the impact of packet loss can sometimes be quite severe for even small rates of packet loss. [5] The second problem is that the alerts by themselves often do not contain enough information to determine whether the attack was successful. The third problem is that these systems are most often implemented with signature-based intrusion detection engines. Signature-based systems may be tuned to produce few false positives; however, they are ineffective at detecting zero-day and advanced persistent threats. [6]

The problem with the second strategy is that lossless compression alone simply is not capable of reducing the amount of traffic enough. Using GNU Zip to compress the 2009 Cyber Defense Exercise data set provides a compression ratio of 2:1. [7] Compression ratios of better than 10:1 are required to minimize the impact of NIDS on day-to-day operations.

The third strategy is to use lossy compression to provide a solution. Network traffic may be considered to be composed of sessions that span spectrums from known to unknown and malicious to benign as illustrated in Figure 3. Quadrant III, the known malicious quadrant, is the domain of intrusion prevention systems as described by Ierace, Urrautia, and Bassett [8]. This research is most interested in quadrant II, the unknown malicious quadrant, because that is the quadrant where evidence of zero-day and advanced persistent threat attacks will be found. In 2004, Kerry Long described the Interrogator Intrusion Detection System Architecture [9]. In this architecture, remotely deployed sensors, known as Gators, collect network traffic and transmit a subset of the traffic to the analysis level. Interrogator employs “a dynamic network traffic selection algorithm called Snapper”. [9]. Long and Morgan describe how they used data mining to discover known benign traffic that they excluded from the data transmitted back to the analysis servers [10]. Smith, Hammell, and Neyens compressed network traffic by removing packets based upon their entropy. [7] Smith and Hammell also truncated packets. [11]

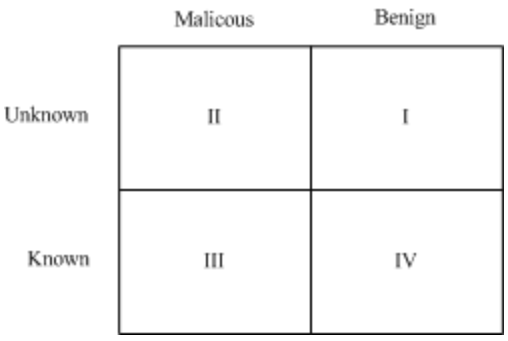

Figure 3. Network traffic composition

3. APPROACH

A transmission control protocol (TCP) session is identified by the internet protocol (IP) address and port of the client and the IP address and port of the server. Each TCP/IP packet contains a source IP address and port and a destination IP address and port. The source and destination change depending on whether the packet was sent by the client to the server or the server to the client. In this paper, a session is considered to be bidirectional and a flow is considered to be unidirectional. This research focuses on flows because it is simple to match source and destination IP addresses and ports. We leave paring opposite flows into a single session for future work.

We began by constructing a tool to read a network capture file in libpcap [12] format and then keep track of the number of packets and bytes in each TCP flow. The flow engine would then return these values as the score for that packet. Packets that scored above the threshold would be dropped. When the flow compression tool completed it would output the number of packets and number of bytes read and written. Snort [3] was then used to analyze the compressed traffic. Upon completion, we extracted the number of alerts detected from the Snort report. We employed the sequence [13] tool to set the thresholds for each run which consisted of several iterations using different thresholds. When the run was completed, we plotted the amount of data transmitted as a percentage of the original, and the alert loss rate (ALR) against each threshold.

Flow Compression Tool

In order to be useful in our compression application the algorithm will need to be implemented as efficiently as possible; therefore, we chose to implement this flow compression tool in C++. A Flow class was implemented which stores the source and destination IP addresses and ports along with the number of packets and bytes observed in the flow. We chose to store these Flow objects in the map container from the standard library. The Flows class contains the map data structure encapsulating that implementation detail from the user. We created a FlowEngine child class of the AnomalyEngine super class which was created in a previous work [14].

As each packet is read from the capture file, it is scored by the FlowEngine. The FlowEngine adds the packet to the Flows object. If this is the first time we have seen this combination of IP addresses and ports, then a new flow is created; otherwise, we increment the packet and byte counters of the existing flow. The flow engine will then return either the current packet or byte count.

Data Sets

In the following section we provide a brief summary of the various data sets that were used in our experiment. It is necessary to abbreviate these data sets. This abbreviation will appear in parenthesis in the text and afterwards appear in tables and captions. Table 1 provides a summary of the duration and packet count for each of these data sets.

DARPA Data Sets: As part of their evaluation of intrusion detection systems, Lippman *et al.* created a data set of synthetic network traffic [15]. We used the testing data from Friday of week 2 (DTE98W2D6). We selected this day because it contains the largest number of alerts in the 2 weeks of testing data.

Cyber Defense Exercise 2009: In 2009 the National Security Agency/Central Security Service (NSA/CSS) conducted an exercise pitting teams from the military academies of the United States and Canada against teams of professional network specialists to see who best defended their

network. Data from this exercise was captured and used by Sangster *et al.* in his efforts to generate labeled data sets [16]. Two network traffic sensors were employed in the exercise: gator-usama010 and gator-usama020. We used the pcapcat [17] program to consolidate the individual hours of for network traffic collected by each sensor into files in Libpcap format. In these trials we used the data captured from gator-usama020 (CDX09U020).

Mid-Atlantic Collegiate Cyber Defense Competition: Based upon the pattern of the Cyber Defense Exercises, a group of industry academics created the collegiate cyber defense competitions [18]. We used the network capture data for the Mid-Atlantic Collegiate Cyber Defense Competitions from 2010 (MACCDC10) which is available from: https://www.netresec.com/?page=MACCDC.

Information Security Centre of Excellence Intrusion Detection System 2012: The University of New Brunswick's Canadian Institute for Cybersecurity created the Information Security Centre of Excellence (ISCX) Intrusion Detection System (IDS) 2012 data set (ISCXIDS12) [19]. This synthetic labeled data set contains full network capture files. We used the data from Monday June 15, 2010. More information about this data set may be found at http://ww.unb.ca/cic/datasets/ids.html.

Canadian Institute for Cybersecurity Intrusion Detection System: The University of New Brunswick's Canadian Institute for Cybersecurity (CIC) created the CIC Intrusion Detection System (IDS) 2017 data set (CICIDS17) [19]. This synthetic labeled data set contains full network capture files. We used the data from Wednesday July 5, 2017. More information about this data set may be found at http://ww.unb.ca/cic/datasets/ids.html.

Real World: We collected real world network traffic from the top level architecture of a laboratory on the Defense Research Engineering Network in Dec 2016 (RW2106).

Table 1. Data sets

Name	Seconds	Packets
D98TEW2D6	90,432	2,177,646
CDX09U020	345,600	42,293,657
MACCDC10	275,666	264,973,151
ISCXIDS12	86,400	34,983,042
CICIDS17	30,458	13,788,878
RW2016	38,337	213,803,423

Rule Sets

The more rules that are tested by Snort, the more computing resources Snort requires to complete its analysis. These resource requirements may climb to the point where Snort is unable to keep up with the network traffic causing packet loss. Therefore, the default rule sets have most of the rules commented out. In order to analyze older data sets, it is necessary to tailor the rule set to ensure that rules appropriate for the time period are active.

Circa2000: The registered Snort rules downloaded from http://www.snort.org in Aug 2013 with rules activated to detected malicious activity from the year 2000.

Circa2009: The registered Snort rules from Aug 2013 with rules activated to detect malicious activity from 2009.

RegAug2013: The registered Snort rules from Aug 2013.

RegSep2018: The registered Snort rules from Sep 2018.

4. RESULTS

We conducted several trials where multiple runs of several iterations of the flow compressor were used to compress the data sets which were then given to Snort for analysis. A script was written to conduct these runs employing the sequence program [13] to define the threshold values. In Tab. 2 we have listed the trial number, data set abbreviation, rule set abbreviation, and the figures where the results are displayed.

For each trial we plot the percentage of the initial size of the data set as the y value and the threshold as the x value as circles. On the same graph we plot the ALR, as the y value and the threshold as the x value in triangles. A table showing the threshold, compression and ALR is provided for each run. These table are used to illustrate the two particular value of interest. The first is the point of most compression with zero ALR. The second is the point of most compression with less than 1% ALR.

Table 2. Experiment Characteristics

Trial	Data set	Rule set	Figures
1	DTE98W2D6	C2000	4 and 5
2	CDX09u020	C2009	6 and 7
3	MACCDC10	RegAug13	8 and 9
4	ISCXIDS12	RegAug13	10 and 11
5	CICIDS17	RegAug18	12 and 13
6	RW2016	RegAug18	14 and 15

Trial 1

The first trial used the D98TEW2D6 data set and the Circa2000 rule set. There were 2 runs where the data set was compressed with a packet threshold. The first used a geometric sequence Eq. (1) where $n = 1$ *to* 20 and the second used a square sequence Eq. (2) where $n = 2$ *to* 32. Figure 4 and Table 3 display the most interesting combined results. Thresholds over 2048 have been omitted because they result in zero alert loss. The data point for threshold 784 was added from the second run because it fills in the gap between 1024 and 512 showing 0.1% packet loss.

$$s_n = 2^n \quad (1)$$

$$s_n = n^2 \quad (2)$$

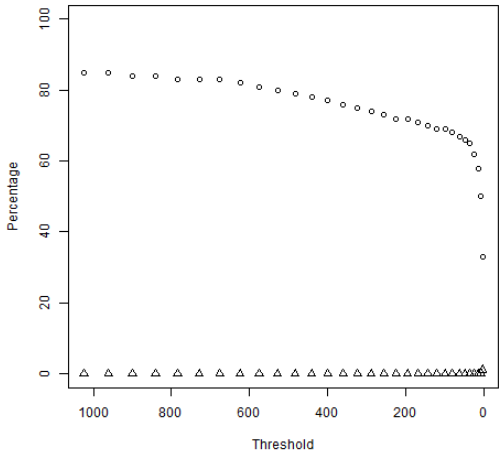

Figure 4. Trial 1 packet threshold

Table 3. Trial 1 packet threshold

Threshold	Compression	ALR
2048	88.00%	0.00%
1024	85.00%	0.00%
784	83.00%	0.01%
512	79.00%	0.01%
256	73.00%	0.03%
128	70.00%	0.04%
64	67.00%	0.05%
32	64.00%	0.08%
16	58.00%	0.13%
8	48.00%	0.33%
4	33.00%	1.16%
2	18.00%	2.25%

There were 2 runs where the data was compressed with a byte threshold. The first used a geometric Eq. (1) sequence where *n = 1 to 20* and the second used a cube sequence Eq. (3) where *n = 3 to 41*. Figure 5 and Table 4 display the most interesting combined results. Thresholds over 131072 were excluded because they also generated zero alert loss. The threshold 343 was added to fill in the gap between thresholds 512 and 256. Thresholds under 128 were excluded because their ALR was over 1%.

$$s_n = n^3 \tag{3}$$

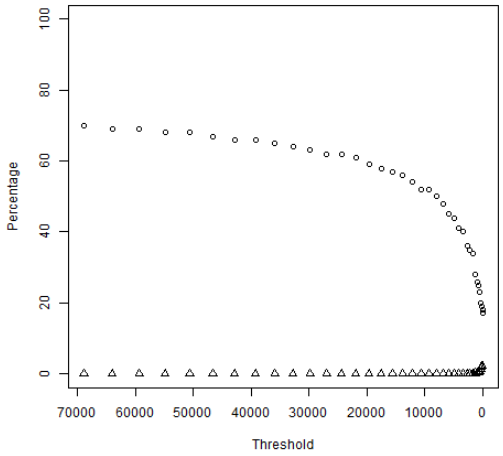

Figure 5. Trial 1 byte threshold

Table 4. Trial 1 byte threshold

Threshold	Compression	ALR
131072	74.00%	0.00%
65536	69.00%	0.01%
32768	64.00%	0.03%
16384	58.00%	0.05%
8192	50.00%	0.08%
4096	41.00%	0.11%
2048	35.00%	0.14%
1024	26.00%	0.29%
512	23.00%	0.58%
343	20.00%	1.27%
256	19.00%	1.29%
128	19.00%	2.25%

Trial 2

The second trial used the CDX09u020 and the Circa2009 rule set. There were 2 runs where the data set was compressed with a packet threshold. The first used a geometric sequence Eq. (1) where *n = 1 to 20* and the second used an arithmetic sequence Eq. (4) where *a = 2*, *d = 2*, and *n = 2 to 61*. Figure 6 and Table 5 display the most interesting combined results. Thresholds over 128 were excluded because they have an ALR of zero. The threshold of 94 and 92 was added from the second run to fill in the gap between the thresholds of 128 and 64.

$$s_n = a + d(n-1) \tag{4}$$

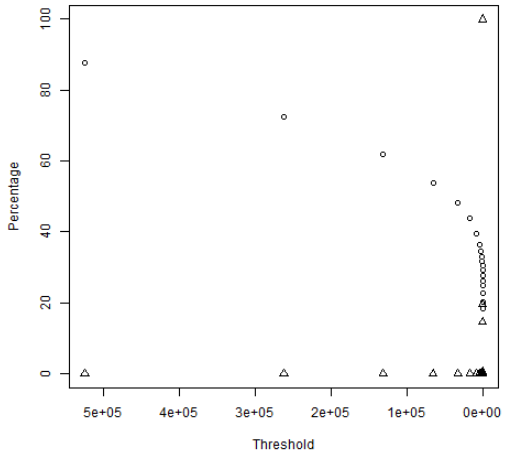

Figure 6. Trial 2 packet threshold

Table 5. Trial 2 packet threshold

Threshold	Compression	ALR
128	29.14%	0.00%
94	28.00%	0.00%
92	28.00%	0.03%
64	27.74%	0.03%
32	26.30%	0.29%
16	24.90%	0.40%
8	22.84%	0.58%
4	20.25%	14.65%

There were 2 runs where the data was compressed with a byte threshold. The first used a geometric sequence Eq. (1) where *n = 1 to 20* and the second used a cube sequence Eq. (5) where n = 19 to 51. Figure 7 and Table 6 display the most interesting combined results. Thresholds over 131072 were excluded because their ALR is zero. The threshold 110592 was added to fill in the gap between 131072 and 65536 and the threshold 6859 was added to fill in the gap between 8192 and 4096.

$$s_n = n^3 \quad (5)$$

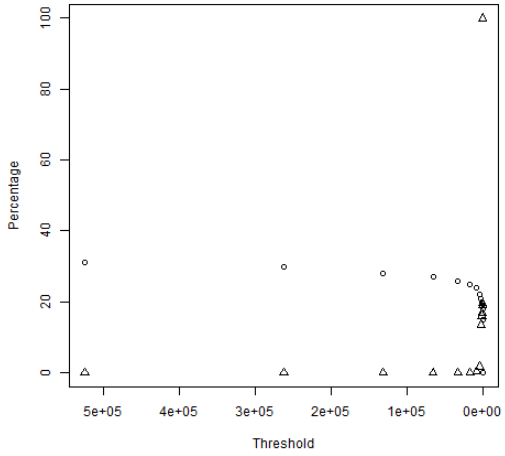

Figure 7. Trial 2 byte threshold

Table 6. Trial 2 byte threshold

Threshold	Compression	ALR
131072	28.00%	0.00%
110592	28.00%	0.00%
65536	27.00%	0.03%
32768	26.00%	0.03%
16384	25.00%	0.09%
8192	24.00%	0.40%
6859	23.00%	0.44%
4096	22.00%	1.75%

Trial 3

The third trial used the MACCDC10 data set and the RegAug13 rules set. There was one run where the data set was compressed with a packet threshold. This trial used a geometric sequence Eq. (1) where *n = 1 to 20*. Figure 8 and Table 7 display the most interesting results. Thresholds over 64 were excluded because their ALR was zero.

There was one run where the data was compressed with a byte threshold. This trial used a geometric sequence Eq. (1) where *n = 1 to 20*. Figure 9 and Table 8 display the most interesting results. Thresholds over 65536 were excluded because their ALR was zero. Although there is a large gap between threshold 66536, which is the last threshold with zero ALR, and threshold 1024, which is the first threshold with an ALR greater than 1%, a second run was not conducted because the size of the file was reduced only 1% at a threshold of 1024.

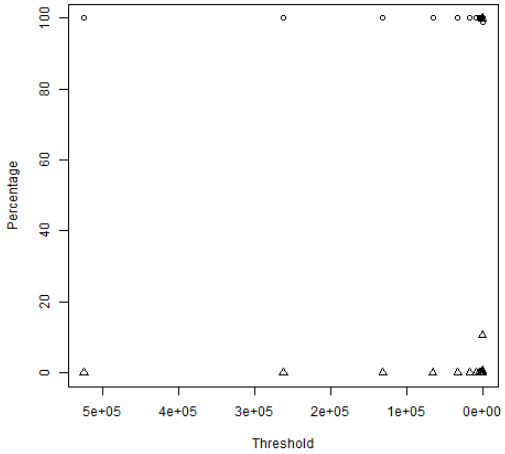

Figure 8. Trial 3 packet threshold

Table 7. Trial 3 packet threshold

Threshold	Compression	ALR
64	99.99%	0.00%
32	99.00%	0.00%
16	99.00%	0.04%
8	99.00%	0.20%
4	99.00%	0.71%
2	99.00%	10.71%

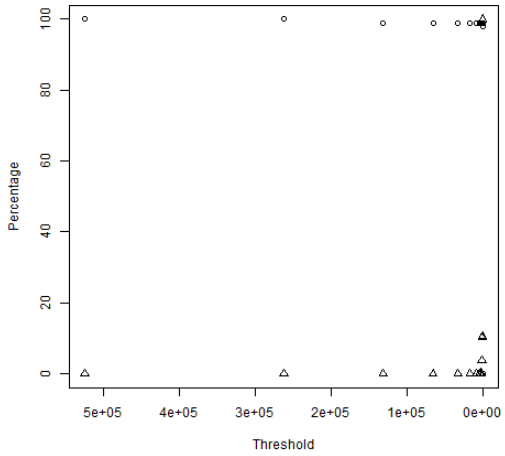

Figure 9. Trial 3 byte threshold

Table 8. Trial 3 byte threshold

Threshold	Compression	ALR
65536	99.00%	0.00%
32768	99.00%	0.00%
16384	99.00%	0.03%
8192	99.00%	0.07%
4096	99.00%	0.11%
2048	99.00%	0.21%
1024	99.00%	3.67%
512	99.00$	10.22%

Trial 4

The fourth trial used the data from June 15th of the ISCXIDS12 data set and the RegAug13 rules set. There were 2 runs where the data was compressed with a packet threshold. The first used a geometric sequence Eq. (1) where *n = 1 to 20* and the second used a cube sequence Eq. (5) where n = 2 to 25. Figure 10 and Table 9 display the most interesting combined results. Thresholds over 16384 were excluded because their ALR was zero. Thresholds less than 4096 were excluded because their ALR was greater than 1%. The threshold 12167 was added from the second run to fill in the gap between 16394 and 8192, and the threshold 5832 was added to fill in the gap between 8192 and 4096.

There was one run where the data was compressed with a byte threshold. This run used a geometric sequence Eq. (1) where *n = 1 to 20.* Figure 11 and Table 10 display the most interesting results. Thresholds over 33554432 were excluded because their ALR is zero. Thresholds under 4194304 were excluded because their ALR were over 1%.

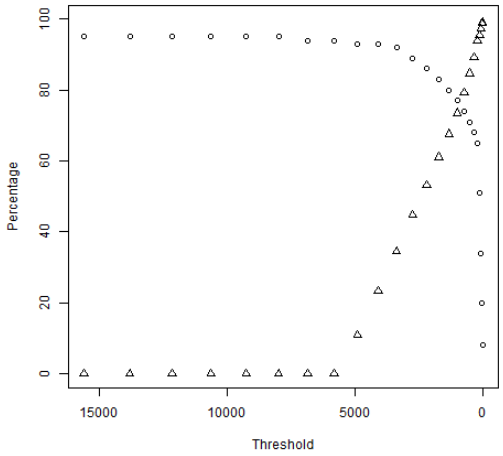

Figure 10. Trial 4 packet threshold

Table 9. Trial 4 packet threshold

Threshold	Compression	ALR
16384	95.00%	0.00%
12167	95.00%	0.00%
8192	95.00%	0.02%
5832	94.00%	0.04%
4096	93.00%	23.43%

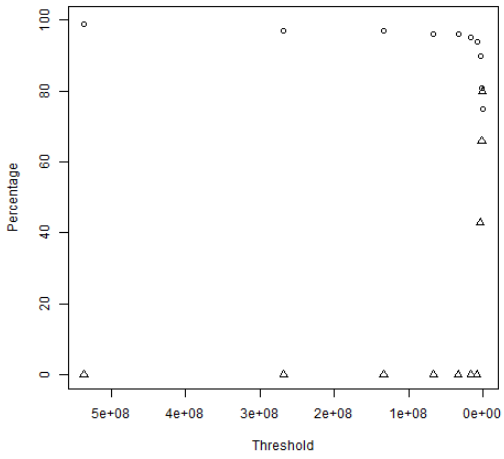

Figure 11. Trial 4 byte threshold

Table 10. Trial 4 byte threshold

Threshold	Compression	ALR
33554432	96.00%	0.00%
16777216	95.00%	0.02%
8388608	94.00%	0.10%
4194304	90.00%	42.95%

Trial 5

The fifth trial used data from day 3 of the CICIDS17 data set and the ReAug18 rules. There were 2 runs where the data set was compressed with a packet threshold. The first used a

geometric sequence Eq. (1) where *n = 1 to 20* and the second used an arithmetic sequence Eq. (4) where *a = 8, d = 1,* and *n = 1 to 24*. Figure 12 and Table 11 displays the most interesting combined results. Thresholds over 128 were excluded because their ALR was zero. Thresholds 31 and 20 were added to fill in the gap between 32 and 16.

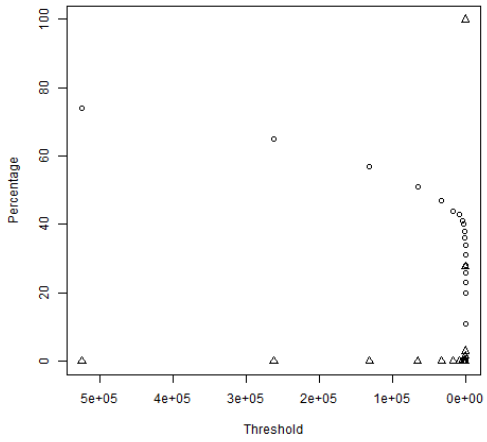

Figure 12. Trial 5 packet threshold

Table 11. Trial 5 packet threshold

Threshold	Compression	ALR
128	31.00%	0.00%
64	28.00%	0.00%
32	26.00%	0.00%
31	25.00%	0.09%
20	24.00%	0.94%
16	23.00%	1.51%
8	20.00%	3.03%

There were 2 runs where the data was compressed with a byte threshold. The first used a geometric sequence Eq. (1) where *n = 1 to 20* and the second used a cube sequence Eq. (5) where n = 16 to 25. Figure 13 and Table 12 display the most interesting combined results. Thresholds over 16384 were excluded because their ALR is zero. Threshold 9261 and 5831 were added to fill in the gaps.

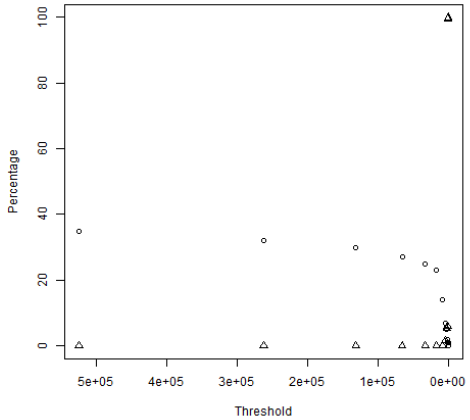

Figure 13. Trial 5 byte threshold

Table 12. Trial 5 byte threshold

Threshold	Compression	ALR
16384	23.00%	0.00%
9261	17.00%	0.00%
8192	14.00%	0.09%
5831	9.00%	0.75%
4096	7.00%	1.51%
2048	5.00%	5.30%

Trial 6

The sixth trial used the RW2016 data set and the RegAug2018 rule set. There was one run where the data set was compressed with a packet threshold. This run used a geometric sequence Eq. (1) where *n = 1 to 20.* Figure 14 and Table 13 display the most interesting results. Thresholds over 32 were excluded because their ALR is zero.

There was one run where the data was compressed with a byte threshold. This run used a geometric sequence Eq. (1) where *n = 1 to 20*. Figure 15 and Table 14 display the most interesting results. Thresholds over 32768 were excluded because their ALR is zero. Thresholds from 1024 to 2 were excluded because their compression and ALR is identical that or thresholds 1024 and 2.

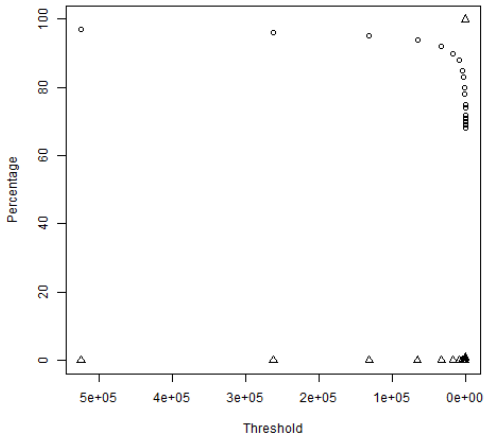

Figure 14. Trial 6 packet threshold

Table 13. Trial 6 packet threshold

Threshold	Compression	ALR
32	71.00%	0.00%
16	70.00%	0.00%
8	69.00%	0.64%
4	68.00%	0.96%
2	68.00%	0.96%

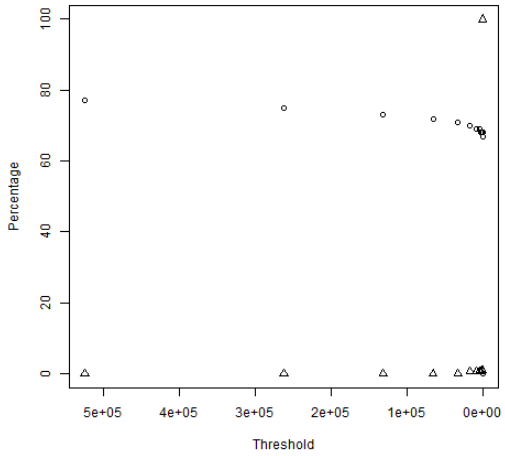

Figure 15. Trial 6 byte threshold

Table 14. Trial 6 byte threshold

Threshold	Compression	ALR
32768	71.00%	0.00%
16384	70.00%	0.64%
8192	69.00%	0.64%
2048	68.00%	0.64%
1024	67.00%	0.96%
2	67.00%	0.96%

Summary

Since the goal is to compress the network traffic as much as possible without losing the ability to detect and investigate malicious activity, Table 15 displays the highest rate of compression for each experiment with no alert loss and the highest rate of compression with less and 1% alert loss.

Table 15. Experiment Results

Trial	Threshold	Value	Size	ALR
1	Packets	784	85%	0.00%
1	Packets	9	50%	0.21%
1	Bytes	121,072	74%	0.00%
1	Bytes	512	23%	0.58%
2	Packets	94	28%	0.00%
2	Packets	8	23%	0.58%
2	Bytes	111,592	28%	0.00%
2	Bytes	6,859	23%	0.44%
3	Packets	32	99%	0.00%
3	Packets	4	99%	0.71%
3	Bytes	32,768	99%	0.00%
3	Bytes	2,048	99%	0.21%
4	Packets	12,167	95%	0.00%
4	Packets	5,832	94%	0.04%
4	Bytes	33,444,432	96%	0.00%
4	Bytes	8,388,608	94%	0.10%
5	Packets	32	26%	0.00%
5	Packets	20	24%	0.94%
5	Bytes	9,261	17%	0.00%
5	Bytes	5,832	9%	0.75%
6	Packets	16	70%	0.00%
6	Packets	8	69%	0.64
6	Bytes	32,768	71%	0.00%
6	Bytes	2	67%	0.96%

5. CONCLUSIONS

The tool performs very differently on different data sets. With the older DTE98W2D6 data set Figs. 4 and 5, compressing by packet thresholds performs poorly while compressing by byte thresholds performs well. With the CDX09U020 and the CICIDS17 data sets Figs. 6, 7, 12, and 13, compressing by packet and byte thresholds perform equally well. With the MACCDC2010 and ISCXIDS 2012 data sets Figs. 8, 9, 10, and 11, compressing by packet and byte thresholds performs poorly. With the RW2016 data Figs. 14 and 15, compressing by packet and byte threshold performed equally poorly.

The difference compressing by packets and bytes on the DTE98W2D6 data sets may be explained by the extensive use of TELNET. TELNET generates a large number of very small packets as each character entered on the keyboard is sent from the client to the server then echoed from the server to the client.

Surprisingly the poor performance of this compression tool on the MACCDC10 and ISCXIDS12 data sets seems to be for opposite reasons. The MACCDC10 data set has a lot of very small flows. This can be seen in that we were able to stop transmitting packets after only 16 while only changing the size of the data set by a single percentage point. These very small flows may have been generated by a large amount of scanning activity. The ISCXIDS12 data set has a lot of very large flows and the malicious traffic seems to be very deep in these flows. This is evident because when we stopped transmitting packets after 12,167, the size of the data set was reduced by 90% but 42.95% of the alerts were lost. The preponderance of alerts were generated in the Hyper Text Transport Protocol. Almost 5600 of these alerts involved a single host, 216.18.165.250. Of these flows 48 of them are over 5000 packets long with as many as 100 alerts per session.

The live capture data is very interesting. We can stop transmitting flows after only 2 packets and compress the traffic to 68% of the original size. The reason behind this is that the flow compressor tool only works on IP version 4 (IPv4) TCP packets. Only 53% of this traffic is IPv4, and only 35% is IPv4/TCP. At 68% compression we have purged almost all of the IPv4/TCP packets. A closer examination of the alerts reveals that only 3 alerts were contained in the TCP traffic. If the flow compression tool handled IP version 6, we would have seen very good performance on this live capture data.

The results of these experiments demonstrate that flows that are malicious manifest that maliciousness early. When we saw malicious activity deep in a flow, it was usually not the first occurrence of malicious activity in that flow. This strategy should be effective in reducing the amount of network traffic sent from the sensor to the CAS. This is especially true when coupled with Snort's ability to capture malicious traffic once it has been detected. Merging traffic compressed by truncating flows with the traffic captured by Snort should provide the analyst with the complete session for review. We achieved some good compression, but flow based compression alone is insufficient to reduce the network traffic that must be transmitted from the sensor to the CAS to less than 10% of the original size.

In future work, we will implement the processing of IP version 6, and conduct experiments with more data sets. The end goal is to integrate this technique and other network

compression techniques including lossless compression to reduce the amount of traffic that needs to be transmitted to the CAS to less than 10% of the initial traffic volume.

6. REFERENCES

[1] S. C. Smith and R. J. Hammell, "Proposal for Kelly Criterion-Inspired Lossy Network Compression for Network Intrusion Applications," *Journal of Information Systems Applied Research*, vol. 10, no. 2, pp. 43-51, aug 2017.

[2] J. L. Kelly, "A new interpretation of information rate," *Information Theory, IRE Transactiosn on*, pp. 185-189, 1956.

[3] M. Roesch, "Snort: lightweight intrusion detection for networks," in *Proceedings of the 13th System Administration Conference (LISA '99)*, Seattle, WA, 1999.

[4] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer Networks*, pp. 2435-2463, 1999.

[5] S. C. Smith, R. J. Hammell, K. W. Wong and J. M. Carlos, "An experimental exploration of the impact of multi-level packet loss on network intrusion detection," in *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*, Towson, MD, 2016b.

[6] R. A. Kremmerer and V. Giovanni, "Intrusion detection: a brief history and overview (supplement to Computer magazine)," *Computer*, pp. 27-30, 2002.

[7] S. C. Smith, S. R. Neyens and R. J. Hammell, "The use of Entropy in Lossy Network Traffic Compression for Network Intrusion Detection Applications," in *Proceedings of the 12th International Conference on Cyber Warfare and Security {ICCWS} 2017*, Reading (UK), 2017.

[8] N. Ierace, C. Urrutia and R. Bassett, "Intrusion Prevention Systems," *Ubiquity*, pp. 2-2, 2005.

[9] K. S. Long, "Catching the Cyber Spy: ARL's Interrogator," Army Research Laboratory, Aberdeen Proving Ground, 2004.

[10] K. S. Long and J. B. Morgan, "Using data mining to improve the efficiency of intrusion detection analsysis," Army Research Laboratory, Aberdeen Proving Ground (MD), 2007.

[11] S. C. Smith and R. J. Hammell, "The use of Snap Length in Lossy Network Traffic Compression for Network Intrusion Detection Applications," *Journal of Information Systems Applied Research*, vol. 12, no. 1, pp. 17-25, 2019.

[12] V. Jacobson, C. Leres and S. McCanne, "PCAP -- packet capture library," 8 March 2015. [Online]. Available: http://www.tcpdump.org/manpages/pcap.3pcap.1.html.

[13] S. C. Smith and R. J. Hammell II, "Controlling Experiments Using Mathematical Sequences," US Army Research Laboratory, Aberdeen Proving Ground, MD, 2018.

[14] S. C. Smith and R. J. Hammell II, "The Use of Packet Header Anomaly Detection in Lossy Network Traffic Compression for Network Intrusion Detection Applications," US Army Research Laboratory, Aberdeen Proving Ground United States, 2018.

[15] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham and M. A. Zissman, "Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation," in *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, Hilton Head, SC, 2000.

[16] B. Sangster, T. O'Conner, T. Cook, R. Franelli, E. Dean, W. J. Adams, C. Morrell and G. Conti, "Toward instrumenting network warfare competitions to generate labeled datasets," in *Proc. of the 2nd Workshop on Cyber Security Experimentation and Test CSET09*, Montreal Canada, 2009.

[17] S. C. Smith, *The effect of packet loss on Network Intrusion Detection. MS Thesis*, Towson, MD: Towson University, 2013.

[18] A. Carlin, D. P. Manson and J. Zhu, "Developing the Cyber Defenders of Tomorrow with Regional Collegiate Cyber Defense Competitions (CCDC)," *Information Systems Education Journal*, pp. 3-10, 2010.

[19] A. Shiravi, H. Shiravi, M. Tavallaee and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357-374, 2012.