Database Techniques for Efficient Network Monitoring and Analysis

Distributed Systems and Write Optimized Databases: Implementations, Trade-offs, and Applications

Evan West
Undergraduate Thesis 2019
University of California, Santa Cruz

***Abstract*—Network connection logs have long been recognized as integral to proper network security, maintenance, and performance management. However, even a somewhat sizable network will generate large amounts of logs at very high rates. This thesis explains why many storage methods are insufficient for providing real-time analysis on sizable datasets and examines database techniques attempt to address this challenge. We argue that sufficient methods include distributing storage and computation, and Write Optimized Datastructures(WOD). Diventi, a project developed by Sandia National Laboratories, is here used to evaluate the potential of WODs to manage large datasets of network connection logs. It can ingest billions of connection logs at rates over 100,000 events per second while allowing most queries to complete in under one second. Storage and computation distribution is then evaluated using Elasticsearch, an open-source distributed search and analytics engine. Then, to provide an example application of these databases, we develop a simple analytic which collects statistical information and classifies IP addresses based upon behavior. Finally, we examine the results of running the proposed analytic in real-time upon bro-conn(now Zeek) flow data collected by Diventi at IEEE/ACM Super Computing 2019.**

I. Introduction

The ability to analyze traffic allows network operators to identify network errors, detect anomalous traffic, classify malware [1], find botnets [2], and more. This analysis can be performed as part of an investigation into a detected breach or as part of a real-time analytic to give network insight or detect zero-day attacks as they occur. In order to enable these applications, the system storing network flows faces many challenges. It must simultaneously store a colossal amount of data, ingest event logs at an extremely high rate, and quickly respond to queries.

The importance of many events is not known at the time the associated log is generated therefore a large amount of data must be stored. The event could be indicative of a misconfiguration or of malicious activity, but the system may not know immediately. This is especially true in the case of security monitoring. In 2018 a majority of successful breaches were not discovered for weeks($<$20%), months(40%) or years(20%). [3] Therefore, forensic analysis of intrusions could necessitate the long term storage of these events.

Additionally, data must be ingested at a high rate in order to keep pace with network events. The system must ingest at least one second's worth of network data in one second so that it does not rapidly fall behind. Queries upon stored logs need to complete quickly so that analytics running in real time get actionable results, to enable rapid automated analysis of past events, and to ensure the sanity of network security operators using the system. Many existing methods for storing network flow fail to manage these competing interests.

Elasticsearch [4] and Diventi [5] are two examples of systems which properly address these concerns, yet the means by which they do are very different. Elasticsearch is a "real-time distributed search and analytics engine." which allows for both rapid ingestion and query response by sharding data across many nodes in a cluster. Diventi on the other hand leverages the write optimizations of a *B to the epsilon tree* (B^ε-tree) to keep up with data ingestion needs while utilizing the underlying B-tree structure to ensure timely queries. This allows Diventi to store a high amount of data on a single node. While we compare Diventi and Elasticsearch here, it is worth noting that they are not mutually exclusive, and both provide unique benefits.

As an example use case of these databases, we created a simple analytic to run through the network history of an IP address. This script produces histograms of the following metrics: number of packets in and out, number of bytes in and out, source and destination port numbers, and number of connections with each neighbor. We utilize these metrics to classify the IP addresses as exhibiting one or more network behaviors.

We discuss methods for storing network flows, then compare and contrast Elasticsearch and Diventi in section II. Then, in section III, current and potential network flow analysis of large flow datasets are discussed. Finally, in section IV, we present the example analytic we developed, the performance of the analytic running in real-time at SC19, and the results we gleaned from this information.

II. Network Flow Databases

A Network flow is a unidirectional stream of packets with common source and destination. Netflow and IPFIX, two common flow export protocols, aggregate packets from this stream within a given window of time into a single flow. Bro-conn logs [6] are not truly flows, as each log refers to a single bidirectional connection which may be composed of multiple packets to open the connection, send data, and

close the connection. All the packets in this connection are aggregated, and for the purposes of this thesis we will refer to bro-conn logs as flows for the sake of simplicity.

Packet aggregation inherently results in the loss of packet specific information. This was an early sacrifice made to address the extremely high volume of events created by logging every packet. However, in spite of this reduction in volume, network flow data still suffers from Big Data complexity. "Big Data is data whose complexity hinders it from being managed, queried and analyzed through traditional data storage architectures, algorithms, and query mechanisms." [7] This complexity is defined by the data's *volume* - the quantity of data to be stored; *variety* - the system must simultaneously hold un-structured, semi-structured, and structured data; and *velocity* - the pace at which data is generated. [7] Thankfully, *variety* is not of concern as network flows all follow a similar structure. Unfortunately, any system tasked with storing this data will still have to contend with high *volume* and *velocity*. This complexity hinders unsophisticated efforts to manage, query, and analyze the data.

The fields of network flows which are commonly used as database keys include the timestamp, originating IP address and responding IP address. This thesis is primarily concerned with queries upon the history of IP addresses and subnets. As such, methods discussed here use IP addresses as keys. While not necessary for these applications, full-text search is certainly an attractive feature as it allows for specific queries searching across multiple fields simultaneously.

A. Naive Methods

The purpose of presenting the following insufficient storage methods is not to make an argument that distributing data or WODs are the only two solutions to this challenge, but rather to further illustrate its complexity. A first attempt to match the speed of the network could be to directly write each log to a file, however querying for a individual log would require a $O(n)$ search through the database. For real-time analysis and responsive queries this is unacceptably slow, especially as n grows to months or years of data.

In response to this setback we might attempt to utilize a data structure such as a hash table to allow a query for a single log to complete in $O(1)$ time. However, this approach fails to take into account *volume*, as it is only effective while a large portion of the stored data can fit within RAM. This limitation of data to RAM is a direct result of the benefits hashing the key normally provides. The avalanche effect, whereby small changes to the text create a large difference in the hash value, ensures that hashed keys are spread across the used storage space. This is an important feature of hashing allowing the hash table to reduce collisions, but inevitably results in an increasing ratio of disk IOs per inserted log. A large number of disk IOs causes rapid degradation of the insertion rate to below the pace of the network. The only solution would be to keep a majority of the data in RAM, but this limits data retention as increasing the size of RAM becomes prohibitively expensive.

NfSen and Flowscan are two common tools used to analyze flows and both suffer from the performance limitations noted above. These methods use a Round Robin Database (RRDtool) [8] [9] to store the data. As a consequence the IP addresses are unordered, and as discussed above must be kept within RAM in order to facilitate quick searches. RRDtool is a time series database which maintains a constant system footprint by automatically overwriting the oldest values with the newest, once the maximum size of the database is reached [10]. However, this severely limits how long network administrators can store potentially important network flows.[1]

Many other potential solutions fall into either of these traps. Elasticsearch and Diventi serve as two examples of how to properly avoid them.

B. Elasticsearch

Elasticsearch uses an inverted index as its underlying data structure [4] to allow data to be stored in the order it arrives while maintaining query performance. An inverted index Fig 1 is a structure which lists all the unique values that appear in any document and the documents in which that value appears. The index can be imagined a map through the unorganized data which allows queries to quickly find what they're looking for. When logging network flows, documents are individual logs and indices include fields such as the originating IP address. Elasticsearch creates an index for each field in the log, allowing operators to query by timestamps, ip addresses, ports, and more.

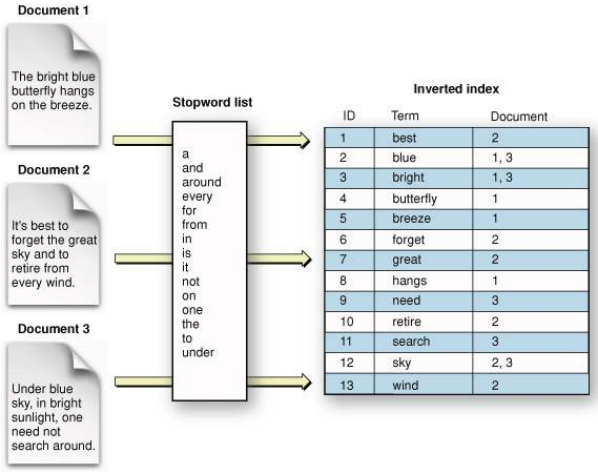

Fig. 1: Constructing an inverted index

When preforming a query, the inverted index returns a list of matching documents which can then be retrieved with minimal search cost. However, one drawback of this structure is that for queries on indices other than time, matching documents will be scattered throughout storage. This means that a single node is unable to take advantage of spatial locality and will

[1]The inclusion of these tools within a section titled 'Naive Methods' is not meant to ruffle feathers. Rather, to illustrate how they are insufficient for the purposes discussed in this thesis.

likely have to contend with a high ratio of memory IOs to access all the documents.

In order to improve data ingestion, query performance, and capacity, Elasticsearch shards its data among many nodes in the cluster. This allows insertions and queries to different shards to run in parallel and additionally allows operators provide the system with more resources without scaling up one device. Scale-up quickly becomes prohibitively expensive, but scale-out is much more cost effective. Data integrity is also ensured by duplicating the data across nodes. If the amount of data stored on each node is not too large, the drawback noted above can now become an advantage to the system as it increases the likelihood that data is found on many shards instead of just one.

C. Diventi

Write Optimized Datastructures are designed to provide efficient write performance at the expense of a limited query performance penalty. A B^{ε}-tree is a B-tree with a insertion buffer placed at each node. Data is inserted to the root buffer which, when filled, flushes its contents to its children Fig. 2.

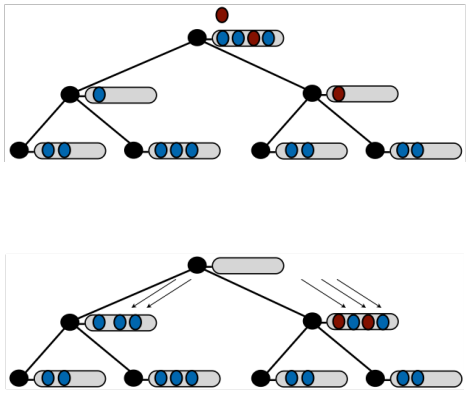

Fig. 2: B^{ε}-tree: Insertion of red data item triggers flush to the child nodes.

This structure has a few key benefits over a B-tree in exchange for paying a small time penalty on queries. The cost of inserting is $O(\frac{\log_B N}{B^{1-\varepsilon}})$, as opposed to a B-tree's cost of $O(\log_B N)$. N is the number of entries in the tree and $B^{1-\varepsilon}$ is the size of the buffer. This may seem to be a small difference in performance, but has much larger implications upon the capacity the database can maintain while keeping pace with the network data. If the system must complete X insertions in one second then the maximum number of logs N is reached when $(\frac{\log_B N}{B^{1-\varepsilon}})X = 1.$[2] $\log_B N$ grows logarithmically with respect to N, therefore, the maximum value of N increases exponentially with respect to the buffer size [5].

[2]This equation is a simplification. There are other costs associated with insertion such as disk access time and various computations. These are normally captured in Big O notation and are forgone here for the sake of simplicity.

As another performance benefit, writes to disk are amortized because buffers higher in the tree are held within caches and RAM. This means that only those flushes which reach the lower levels of the tree trigger blocking disk IOs. This increases the rate of ingestion. The time penalty to query performance is a consequence of the need to search through the buffer of each node visited while traversing the tree. This means that there is a positive correlation between buffer size and query latency. For more details see Raizes et al. [5].

Diventi orders data first upon the source ip addresses and then the timestamp. The benefit of this is that queries to IP addresses and subnets are quick. The logs which match the query will be contiguous within the database and quick to identify as a result of the B-tree structure. The system can take advantage of spacial locality when performing these queries in addition to quickly identifying the matching logs. As such, disk IO penalties during queries should be minimized. The drawback is that the system can not efficiently query on any other field, however, the workload we are concerned with is primarily investigations into individual ip addresses or subnets. Another drawback is that in order to retrieve logs when the queried IP address matches either the source or responding IP address, two logs of each event must be inserted. One with normal IP ordering and the other reversed.

Diventi provides efficient use of resources to allow a large amount of data to be stored on a single node while maintaining high performance. Multiple Diventi nodes responsible for different network taps or for data that is split between them by another process may be deployed if even more capacity is required.

III. Network Flow Analysis

Network flows do not contain packet payloads information[3], and do not provide packet level granularity for fields such as the number of bytes per packet or TCP flags. Instead, flows contain aggregate totals of the number of packets, bytes, flags used in any packet, and more. Despite this loss of specifics, flows still contain sufficient information to identify network intrusions [11]. This section describes existing and potential network flow analysis to be improved or enabled by network flow databases.

A. Current Methods and Research

1) Blacklist Approach: At the most basic level Nfsen and Flowscan (performance discussed in subsection II-A) provide for analysis of flow statistics and traffic. They create statistical summaries and graphical displays of data in addition to providing the ability to filter results by a variety of fields. Nfsen additionally provides the ability to define alerts. These alerts can act upon a filtered subset of the overall traffic, trigger on up to 6 chained conditions, and take a set of actions [8]. Similarly the Bro-IDS provides the ability to trigger actions, such as blocking traffic and creating alerts, based upon the

[3]Even in the case where payloads are stored, payload analysis is becoming increasing untenable as encryption becomes more ubiquitous.

content of the packets[4] collected by the tap [6]. This is often accomplished using IP blacklists and checking packet contents against common malware patterns.

2) Zero-day attacks: Statistical analysis and machine learning are employed for detecting and classifying a wide variety of network traffic patterns. However, they commonly share the goals of reducing the false positives generated by systems like those discussed above, and detecting zero-day attacks.

There are many examples of using network flows to detect and classify actions taken by network participants. Moustafa et al. present an ensemble based technique for detecting exploits of IoT systems, particularly botnets, using statistical summaries provided by the Bro-IDS [2]. *MalClassifier*, a tool developed by researchers at Oxford, uses the network flow behavior of malware to classify it into various malware families without requiring sandbox execution [1]. *MalCalssifier* additionally has the ability to determine if the malware does not fit previously established malware families, allowing security operatives to propose new families. Finally, Rodriguez et al. present work in using time series databases studying historical patterns to predict future behavior and detect anomalies. They state that the more data that is used in the time series the more accurate the predictions will be.

B. Potential Applications

This portion of the thesis attempts to address the possible uses of a fully operational Elasticsearch or Diventi database.

1) Lateral Network Movement: Lateral network movement is a process by which an attacker takes advantage of access to one machine in the network to gain access to another machine. This is done for the purpose of reconnaissance to find future targets, to reach an objective, or gain a higher level of access to the network [12].

Detecting how a bad actor or piece of malware has moved within one's network is essential for a proper response to an intrusion. Otherwise, malware may remain within the network, continuing to cause damage after action has been taken to address the compromise. Additionally, this type of monitoring may allow security operators to detect anomalies. A chain of ssh logins may be indicative of an attack. In order to identify this movement, it is necessary to hold a large amount of network data. This requirement necessitates the use of a proper network flow database. Additionally, sensors that monitor local traffic are required. The more network visibility the system is given the better, if the system has no view of the connection between computer A and B than it cannot detect lateral movement between them.

Tracking lateral movement was considered for this thesis but was ultimately forgone as a result of limited network visibility.

2) Machine Learning, Human Interaction and Verification: The current work presented in subsection III-B is useful in detecting anomalous network behavior and zero-day attacks. However, for complicated use cases, a human security operator will likely have to interact with the machine learning algorithm to verify that the correct actions were taken or to interpret results. To facilitate this it may become necessary for the user to look into the history of ip addresses which the algorithm has flagged. These queries by the human operator need to complete quickly and have access to a large amount of data in order to facilitate the interaction and save valuable analyst time.

[4]The Bro-IDS analyses traffic on the wire in addition to generating logs. This allows it access to the contents of each packet.

IV. IP Flow Analysis at Super Computing

To begin to evaluate the performance and use of a network flow database, we developed an analytic to produce flow metrics in real-time at SC19. Diventi indexed bro-conn logs from a tap used by the SciNet Security Team over the course of the event. At the time the analytic was run, Diventi had indexed a quarter of a billion events corresponding to half a billion logs. What follows is a description and evaluation of the analytic.

A. Description

The analytic is designed with the goal of gathering basic flow statistics about an IP address. We collected the total number of connections, number of packets in and out, number of bytes in and out, source and destination port numbers, and neighbors. Diventi records the magnitude of the packets and bytes by storing $\log_2 x$ rather than the exact number x. For example, records with byte counts between 1 and 3 are recorded using magnitude 1 and records with a byte count from 4 to 7 recorded with magnitude 2. The number of packets and bytes is therefore already bucketed so as to obscure small differences and highlight large ones. The information was collected in real time and then processed to see what basic conclusions we could reach from the data. We classify each IP address as follows:

1) *Active* if it has more than 100 connections within our network and *inactive* if it has less than 20.
2) *High degree* (number of neighbors) if the degree is greater than 30 and a *small degree* if less than 5.
3) *Receiver* if it receives twice the number of packets it sends and a *sender* if the opposite is true.
4) *Elephant* if the average number of bytes sent or received per connection is greater than $10,000$ bytes, a *mouse* if both are less than 1000 but greater than 80, and a *gnat* if both are less than 80.

These categories were defined somewhat arbitrarily with the goal of demonstrating the ability to quickly categorize the behavior of an IP address. As discussed earlier, deep inspection into the history of IPs or subnets is the purpose of this work.

B. Query Performance

We first discovered that the performance of queries across a range of logs was fairly constant even as the size of the database grew much larger. We posit that this is because the majority of a query workload is composed lateral scans through the leaves of the tree. Therefore, increases in the smaller cost to traverse down the tree and find the first

matching key are relatively insignificant. Especially as the cost of traversal increases as the log of the number of records.

We show this trend in Table I. 1 million logs were queried at intervals when the database had stored between 1 million and 1 billion logs. Query latency increased by 20% while the size of the database increased by nearly 1000% over the same period, showing the relatively flat latency. In order to ensure uniform results, the query time is an average of three queries. The server was shut down between each query to prevent the results from being cached. A Dell PowerEdge R520 with 165GB of RAM and 32 cores was used for this test, however, the size of the RAM was not a significant factor, as Diventi does not preemptively load data from the underlying storage into its cache.

TABLE I: Query Performance Versus Size of Database

Stored Logs (millions)	Query Latency (s)[a]
1	1.947013
100	2.058103
200	2.167957
500	2.208140
1000	2.337618

[a]Serverside Latency, averaged over 3 queries.

Table II shows the performance of the metrics analytic running upon Diventi at Super Computing. The Unix utility *time* was used to measure the total amount of time required to create the metrics on the client's end. To establish the performance and generate data we queried a single IP address, a 255.255.255.0 subnet mask, a 255.255.0.0 subnet mask, and the entire database. *Real* refers to the total time from the start of the program to completion, *User* to the amount of time the process was executing on the CPU, and *System* to the amount of time the process was executing system calls. We see an increasing amount time spent off the CPU in Table II likely because of the increasing size and complexity of the data stored on the client end, causing IO blocking when preforming analysis.

TABLE II: Analytic Performance

Description	Logs Processed	Time to Complete (s)		
		Real	User	System
Single IP	1,005	0.079	0.024	0.028
Subnet \8	32,864	0.425	0.100	0.076
Subnet \16	372,755	2.386	0.936	0.336
Everything	485,997,746	104 min	22 min	16 min

The analytic was able to very quickly establish the statistical history of an ip address or subnet by taking advantage of the B^ε-tree's structure. This provides evidence that network flow databases will allow complex analysis to complete rapidly.

C. Results

Fig. 3 shows graphical representations of the metrics collected upon a single IP address. This IP had 603 connections, the source port was scattered, but the destination port was always 13568. From this information the IP address is classified as active, of small degree, neither a sender nor receiver, and a mouse. This IP address was likely receiving and sending data to a small number of other IP addresses from a process running on port 13568. The neighbor histogram indicates that the behavior of this IP address is likely most dependent upon neighbor 4.

At the time the analytic was run 1,480,024 IP addresses were stored within the database. Based upon the statistical summary returned when the analytic was run across the entire database each IP address was matched with the classifications described in subsection IV-A. The number of IP addresses that match each category are shown in Table III along with the percentage of the total IPs which matched. Using this table it is observed that a vast majority of IPs were classified as inactive, small degree, senders, or gnats. Based upon this information, we could posit that a vast amount of the traffic collected was composed of simple interactions which didn't require much data to be transferred back to the receiver therefore most packets were sent by the originator. Some examples of this type of traffic include DNS lookups, ICMP messages, and SYN scanning.

TABLE III: Classifications of Super Computing Traffic

Classification	Number of Matches	% of Total
Inactive	1,176,097	79.46
Active	128,218	8.66
Small degree	995,741	67.28
High degree	204,491	13.82
Senders	1,187,979	80.27
Receivers	77,541	5.24
Elephants	14,789	1.00
Mice	94,610	6.40
Gnats	1,348,347	91.10

D. Limitations

First, it should be noted the analytic provides no ability to determine the portion of IPs which matched multiple categories. Each set of classifications: inactive and active, small degree and high degree, etc. is calculated in isolation. A simple improvement to the analytic would be to add this capability allowing the user to zero in on particularly rare behaviors.

Diventi was granted only limited access to the Super Computing network. As a consequence of this limited network visibility, some results may be incomplete. It is important that any organization seeking to employ network monitoring carefully consider the implications of the visibility provided by their network taps.

V. Conclusion

Using databases designed for the big data challenges associated with logging and querying network flows is necessary in order to provide network operators with a larger, more efficient window into the network traffic of both past and present. Its imperative for the development of automated analytics and for effective post-intrusion investigations that these methods are

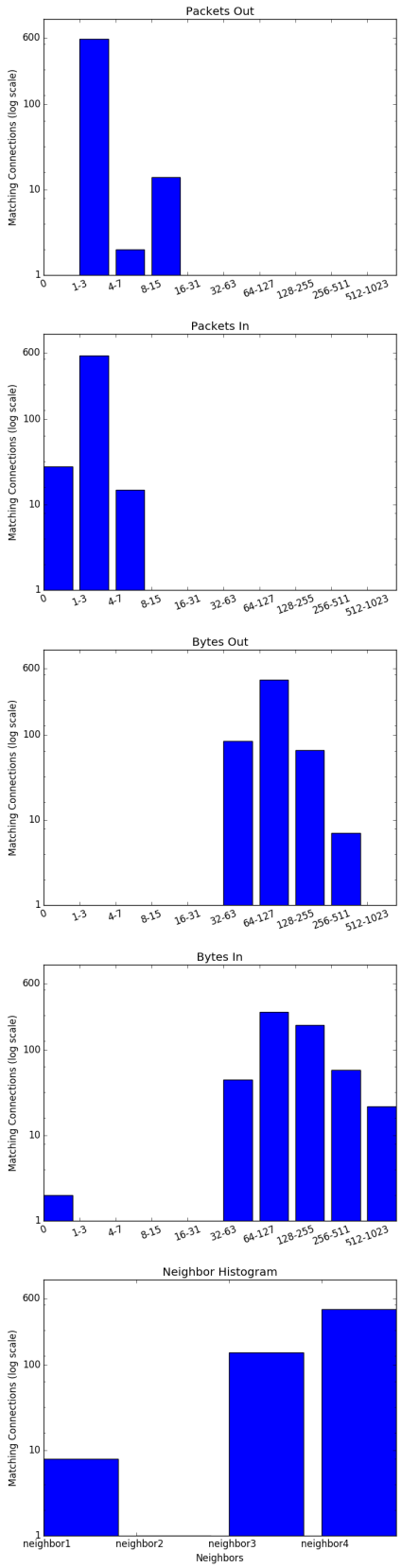

Fig. 3: Metrics for single IP address. Y-axis in log scale

adopted when logging network flows. This thesis shows that Diventi was able to ingest over a billion logs while providing rapid query responses with relatively flat latency. These queries are capable of quickly collecting information regarding IP addresses to classify them into various categories. Running this analysis at Super Computing 2019 revealed that a majority of the traffic collected was composed of simple interactions such as DNS lookups, ICMP messages, and SYN scanning. In conjunction with machine learning and other cutting edge techniques, these databases allow security personnel to use their time to efficiently identify threats and respond to alerts instead of waiting for information.

Acknowledgment

I would like to thank the following people who made this thesis possible. Peter Alvaro and Katia Obraczka for serving on my thesis committee, for their feedback on the drafts of this thesis, and for their invaluable instruction. Thomas M. Kroeger, my mentor at Sandia National Laboratories, for his assistance in this thesis and guidance over the years. Finally, my friends and family for the support they gave me through my undergraduate journey. I can't wait for the next adventure. Don't forget your towel[5].

References

[1] BA. AlAhmadi, and I. Martinovic "MalClassifier: malware family classification using network flow sequence behaviour." 2018 APWG Symposium on Electronic Crime Research, p. 1-13, IEEE 2018.

[2] N. Moustafa, B. Turnbull, KR. Choo "An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things." IEEE Internet of Things Journal, vol. 6, 2019.

[3] 2019 Data Breach Investigations Report [Online] Available: https://enterprise.verizon.com/resources/reports/2019-data-breach-investigations-report.pdf.

[4] Elasticsearch: The Definitive Guide [Online] Available: https://www.elastic.co/guide/en/elasticsearch/guide/master/index.html

[5] J. Raizes, E. West, TM. Kroeger, B. Wight, C. Phillips, J. Berry, M. Bender, and R. Johnson. "Tracking network events with write optimized data structures." Network Research Exhibition, SuperComputing 2018. in press.

[6] Zeek Network Security Monitor [Online] Available: https://www.zeek.org/.

[7] T. Mahmood, and A. Uzma. "Security analytics: Big data analytics for cybersecurity: A review of trends, techniques and tools." 2nd national conference on Information assurance, p. 129-134, 2013 IEEE.

[8] P. Haag Nfsen: Netflow sensor [Online] Available: nfsen.sourceforge.net.

[9] D. Plonka Flowscan [Online] Available: www.caida.org/tools/utilities/flowscan/.

[10] T. Oetiker, J. Brutlag, and A. Bogaerdt RRDtool [Online] Available: https://oss.oetiker.ch/rrdtool/.

[11] A. Sperotto, R. Sadre, and A. Pras "Anomaly characterization in flow-based traffic time series." International Workshop on IP Operations and Management. p. 15-27 Springer, Berlin, Heidelberg, 2008.

[12] P. Neise "Graph-based event correlation for network security defense." PhD diss., The George Washington University, 2018.

[13] AC Rodriguez, MR de los Mozos "Improving network security through traffic log anomaly detection using time series analysis." Computational Intelligence in Security for Information Systems 2010, p. 125-133, Springer, 2010

[5] A towel, [...] is about the most massively useful thing an interstellar hitchhiker can have.