Application of Genomic Compression Techniques for Efficient Storage of Captured Network Traffic Packets

By

James A. Loving

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

In

Information Assurance

College of Engineering and Computing
Nova Southeastern University

October 2023

We hereby certify that this dissertation, submitted by James Loving conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

[illegible]

James D. Cannady, Ph.D. **Chairperson of Dissertation Committee**	**10/30/23** **Date**

[illegible]

Ling Wang, Ph.D. **Dissertation Committee Member**	**10/30/23** **Date**

Sumitra Mukherjee

Sumitra Mukherjee, Ph.D. **Dissertation Committee Member**	**10/30/23** **Date**

Approved:

Meline Kevorkian

Meline Kevorkian, Ed.D. **Dean, College of Computing and Engineering**	**10/30/23** **Date**

College of Computing and Engineering
Nova Southeastern University

2023

An Abstract of a Dissertation Submitted to Nova Southeastern University in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Application of Genomic Compression Techniques for Efficient Storage of Captured Network Traffic Packets

by
James A. Loving
October 2023

In cybersecurity, one of most important forensic tools are audit files; they contain a record of cyber events that occur on systems throughout the enterprise. Threats to an enterprise have become one of the top concerns of IT professionals world-wide. Although there are various approaches to detect anomalous insider behavior, these approaches are not always able to detect advanced persistent threats or even exfiltration of sensitive data by insiders. The issue is the volume of network data required to identify this anomalous activity. It has been estimated that an average corporate user creates a minimum of 1.5 MB audit data per day or roughly 30 MB per business month and thus 90 MB or more in a three-month period. That volume by itself is not unwieldy, but that is for a single user. If a large corporate network is involved, that number could easily reach one-half petabyte or more, a size that could be unwieldy to store for any length of time. Normal compression techniques can reduce this size significantly, but the resultant file not only may still be large, but it also requires decompression to its original size to analyze.

In gene research, file size is also a major concern. An approach has been developed whereby common segments of a gene are stored as links to the original, augmented with edit scripts showing any difference, such that the resultant file is significantly smaller than the original, allowing for easier analysis. The purpose of this research was to apply dynamic compressive techniques utilized in genomic research to the issue of data volume. All available data is required for gene sequencing, so compressive techniques have been developed where redundant information is replaced by links to that data, leaving only the difference intact for analysis. Similarly, in this research, network traffic was processed such that redundant packet information was replaced by links to that information, leaving intact the pertinent information needed to reconstruct the packet information and the steps required for the access.

To test the Genomic Network Compression System (GNCS), two datasets were chosen, packet captures from the 2012 Mid-Atlantic Collegiate Cyber Defense Competition (MACCDC) and a hybrid dataset from the University of New South Wales at the Australian Defense Force Academy, Canberra, Australia, the UNSW-NB15 17-2-2015 dataset. To test for the efficacy with request/reply message formats, Address Resolution Protocol packets were processed for both datasets and obtained file size savings of 54.8% and 49.6% respectively. To test the GNCS with protocols that transfer large amounts of data, the Transmission Control Protocol was processed for both datasets. The MACCDC 2012 dataset consistently exhibited file space savings of approximately 66%, while the

UNSW NB-15 dataset showed a gradual increase from 10.3% for a sample of 1,000 packets and increased until it plateaued at approximately 46% for samples of 10,000,000 packets and larger. This shows that the GNCS can provide approximately a 50% savings in storage space for network packets, providing organizations with a significant decrease in the required storage space for audit files.

Acknowledgements

First and foremost, I want to acknowledge the patience and support over these many years from my wife, Dottie. Through the frustration and exasperation of my lingering research, she stood there giving me the support to go on. I would also like to acknowledge my children and their loved ones, whose support and encouragement also gave me the strength to continue. Finally, I must acknowledge my dissertation committee for their time and guidance. In particular, I want to thank Dr. Cannady, whose patience and guidance through my long journey were key to my success.

Table of Contents

Abstract iii
List of Tables viii
List of Figures x

Chapters

1. **Introduction** **1**
   - Background 1
   - Problem Statement 5
   - Dissertation Goal 6
   - Relevance and Significance 7
   - Barriers and Issues 9
   - Assumptions, Limitations, and Delimitations 10
   - Summary 11

2. **Review of the Literature** 13
   - Overview 13
   - Context 14
   - Audit Log Files 15
   - Intrusion Detection 19
   - Network IP Flow Techniques 21
   - Compression Techniques 22
     - Lossless Compression 23
     - Lossy Compression 24
     - Compressed Bitmap Indexes 26
     - Network Compression 30
     - Genomic Data Compression 32
     - Approximate Matching 36
   - Summary 38

3. **Methodology** 40
   - Overview 40
   - Development Considerations 42
   - Development Resources 47
   - Dataset Analysis 48
   - Parser Development 51
     - Internet Layer Parser 53
     - ARP Parser 54
     - IPv4 Parser 55

Transmission Control Protocol (TCP) Parser 55
User Datagram Protocol (UDP) Parser 59
Compression Algorithm 60
Output File structure 70
Data Analysis 72
Summary 73

4. **Results** 75
Introduction 75
Parser Development 76
Arp Parser 76
Complete ARP Pair 78
Orphan ARP Packets 79
Duplicate ARP Packets 79
IPv4 Parsers 79
Transmission Control Protocol (TCP) Parser 80
User Datagram Protocol (UDP) Parser 84
Genomic Compression Algorithm 86
Final Program Results 90
Arp Results 90
TCP Results 91
Summary 94

5. **Conclusions, Implications, Recommendations, and Summary** 97
Conclusions 97
Implications 101
Recommendations 102
Summary 104

Appendices 111
A. NIST SP 800-53 Revision 5, Audit and Accountability Security Controls 111
B. Dataset Analysis and Selection 114
C. MACCDC Datasets 129
D. UNSW-NB15 Datasets 135
E. Application Layer Protocols 141
F. Process Results 149
G. Acronyms 152

References 155

List of Tables

Tables

1. Processing Time Comparison 67
2. Transport Layer Protocols..86
3. ARP Results 90
4. UNSW-NB15 Dataset Output..100

Appendix A – NIST SP 800-53 Rev5 Audit and Accountability Security Controls.111

A1. NIST SP 800-53 Revision 5 Audit and Accountability Security Controls..............111

Appendix B – Dataset Analysis and Selection 114

B1 Candidate Datasets...114
B2 Candidate Datasets: File and Packet Statistics – MACCDC Datasets......................115
B3 Candidate Datasets: File and Packet Statistics – UNSW-NB15 Datasets 116
B4 Candidate Datasets: File and Packet Statistics – MILCOM Datasets.......................118
B5 Candidate Datasets: File and Packet Statistics – MILCOM Datasets (Cont.)...........119
B6 Candidate Datasets: File Header Extracts..122
B7 Candidate Datasets: Internet Layer Protocols..123
B8 Candidate Datasets: Transport Layer Protocols..125

Appendix C – MACCDC Datasets 129

C1 PCAP File Summaries MACCDC 2010 Dataset..129
C2 PCAP File Summaries MACCDC 2011 Dataset..130
C3 PCAP File Summaries MACCDC 2012 Dataset..130
C4 Internet Layer Protocols MACCDC 2010 Dataset 131
C5 Internet Layer Protocols MACCDC 2011 Dataset 131
C6 Internet Layer Protocols MACCDC 2012 Dataset 132
C7A Transport Layer Protocols - MACCDC 2011 Dataset – Top 5.............................133
C7B Transport Layer Protocols - MACCDC 2011 Dataset – Details...........................133
C7A Transport Layer Protocols - MACCDC 2012 Dataset – Top 5.............................133
C7B Transport Layer Protocols - MACCDC 2012 Dataset – Details...........................134

Appendix D – UNSW-NB15 Datasets...135

D1 PCAP File Summaries UNSW-NB15 17-2-2015 Dataset..135
D2 PCAP File Summaries UNSW-NB15 22-1-2015 Dataset..136

D3 Internet Layer Protocols UNSW-NB15 17-2-2015 Dataset 137
D4 Internet Layer Protocols UNSW-NB15 22-1-2015 Dataset 138
D5A Transport Layer Protocols - UNSW-NB15 17-2-2015 – Top 5.............................139
D5B Transport Layer Protocols - UNSW-NB15 17-2-2015 – Details...........................140
D5A Transport Layer Protocols - UNSW-NB15 22-1-2015 – Top 5.............................140
D5B Transport Layer Protocols - UNSW-NB15 22-1-2015 – Details...........................140

Appendix E – Application Layer Protocols..141

E1A TCP Application Layer Protocol Occurrences – MACCDC 2012 142
E1B TCP Application Layer Protocol Details – MACCDC 2012 – Top 5....................142
E1C TCP Application Layer Protocol Details – MACCDC 2012 – Next 5 143
E2A TCP Application Layer Protocol Occurrences – UNSW-NB15 17-2-2015...........143
E2B TCP Application Layer Protocol Details – UNSW-NB15 17-2-2015 – Top 5......144
E2C TCP Application Layer Protocol Details – UNSW-NB15 17-2-2015 – Next 5 144
E3A UDP Application Layer Protocol Occurrences – MACCDC 2012........................145
E3B UDP Application Layer Protocol Details – MACCDC 2012 – Top 5 146
E3C UDP Application Layer Protocol Details – MACCDC 2012 – Next 5..................146
E4A UDP Application Layer Protocol Occurrences – UNSW-NB15 17-2-2015..........147
E4B UDP Application Layer Protocol Details – UNSW-NB15 17-2-2015 – Top 5 147
E4C UDP Application Layer Protocol Details – UNSW-NB15 17-2-2015 – Next 5....148

Appendix F – Process Results...149

F1 MACCDC 2012 Dataset Results 150
F2 UNSW-NB15 22-1-2015 Dataset Results 151

List of Figures

Figures

1. Lempel-Ziv Algorithm Family 23
2. ARP Packet Format....................77
3. TCP Segment Format....................81
4. User Datagram Protocol....................84
5. MACCD Data Distribution....................94
6. UNSW-NB15 Data Distribution....................94

Chapter 1

Introduction

Background

Cybersecurity professionals agree that it is essential to collect and maintain audit files (audit trails) in order to ensure accountability, determine who is accessing a network, the applications they are using, any activity that could impact the security posture of the enterprise, and allow for reconstruction of events after a problem has occurred (Marker, 2021; NIST ITL Bulletin, 1997; Shopp, 2020). Operationally, enterprises utilize various intrusion detection systems, as well as Security Information and Event Manager (SIEM) systems which collect and correlate security data from multiple devices, report the results, and raise appropriate events (Mokalled, H., 2019). The actual functioning of these types of network systems was beyond the scope of this paper. Rather, the data streams and files these and other enterprise devices utilize was the focus – audit files and network traffic archive files stored over a long term to analyze for malicious activity.

Audit logs can be extremely varied in their content, depending upon the source of the log and its intended use. Most log files document events that have happened, the content is limited, and follows some documented guidance, such as that provided by the National Institutes of Science and Technology (NIST) Special Publications (SP) 800-92 Guide to Computer Security Log Management. However, some logs, such as those required for intrusion detection, require more information or even aggregated data flows to detect unauthorized activities. Network intrusion detection systems can perform packet captures to analyze the entire connection should an alert occur (NIST SP 800-94, 2007). It is this latter form of audit logging that will have the tendency to become large, since the entire contents of the Ethernet packet, including the data segments, must be maintained, and not just the limited fields maintained in event-based audit logs.

To address processing large amounts of data in real-time or near real-time, some researchers have turned to distributed processing (Forestiero, 2015; Sindhu, Ramasubramanian, and Kannan, 2004) or modular neural networks (Golovko, 2007). In this manner, they could monitor heavy network traffic with a minimal impact on throughput. Although all these approaches address the real-time detection objective, they do not allow for long-term storage for future correlation and reconstruction.

Depending upon the source of an audit log file, the size of the files can get quite large. In a presentation at the 2015 ACM Conference on Data and Applications Security (CODASPY) discussing use of SIEM tools at Hewlett Packard (HP), William Horne indicated that their Domain Name Service (DNS) clusters process approximately 16 billion packets per day, or more than 5.8 trillion packets per year (Horne, W., 2015). The HP presentation was limited to their DNS servers. Typical DNS communications occur in pairs – a query and a response (DNS, 2022). Although a DNS query or response format allows for multiple queries/responses per message, the overall query transaction size is relatively small and would fit within a single Ethernet packet (The TCP/IP Guide, 2005). Therefore, the 16 billion packets would relate to approximately 8 billion query/response pairs. A packet flow is defined as the total sequence of packets between a source computer and its destination. If the discussion is limited to the Internet Protocol (IP), there will be two sets of packet flows, from the computer initiating traffic to the destination and then another packet flow for the return communications (Kerner, 2021). In a paper by Kim, et. al., (2004), the average flow contained 28 packets, comprised of 18,239 bytes. If we assume that each DNS query is a prelude to an internet connection and thus a packet flow, then the resultant traffic would equate to approximately 3,700 TB per day. To take a very conservative approach, we will assume that 90% of the traffic is for external users accessing public internet pages, thus decreasing the traffic of interest to perhaps 370 TB per day. For the US Department of Defense, sources and methods intelligence (SAMI) audit records must be maintained for five (5) years, and all other audit records retained for one (1) year[1]. If we then also assume activity will occur on business days only, this will equate to over 90 petabytes of network traffic annually, which may need to be maintained for audit purposes.

The NIST ITL Bulletin, (1997) further discusses the value of audit logs in reconstructing events after a problem has occurred, by using the audit log to reconstruct the series of steps taken by the systems, users, and applications leading up to the event. (The NIST ITL Bulletin, 1997). The results of the HP presentation and the private study above are representative of the underlying issue: how do you maintain large data files for future analysis?

[1] These time periods are obtained from https://www.stigviewer.com/controls/8500/ECRR-1 that covers a security control from the DoD Information Assurance Certification and Accreditation Process (DIACAP) security catalog. DoD has subsequently transitioned to the Risk Management Framework. Control ECRR-1 has been replaced by the RMF security Control AU-11. However, access to the new DoD security control parameters requires a valid DoD-issues Computer Access Card (CAC). Therefore, the older, publicly accessible URL parameters are cited.

To address the problem of storing large amounts of data, various file compression techniques have been developed. Although formats such as ZIP, RAR, and TAR are commonly thought to be compression techniques, they are archiving formats that include various compression techniques for efficient archiving (FileInfo, n.d.).

In a review of compression methods, Kavitha (2016) identified two general categories of compression: Lossless and Lossy compression. Lossless compression is mainly used for archiving and includes those methods in which after decompression, the resultant file is identical to the original, bit for bit; some of the main techniques include Run-Length Encoding (RLE), Lempel-Ziv-Welch (LZW), and Huffman encoding. Lossy compression is used where loss of some data is determined not to impact the essential information of the original; the main lossy compression techniques include JPEG, MP3, and MP4 (Kavitha, 2016). In 2018, Uthayakumar, Vengattaraman, & Dhavachelvan discussed the common compression techniques based on data quality, coding scheme, data type, and application. Network traffic is a combination of all possible data types that are represented by a stream of 16-bit hexadecimal characters. The final compression ratio will be dependent upon the mixture of the data – the more text, the higher the possible compression ratio. Regardless of the technique utilized, standard compression techniques create static compressed files. To access the data to perform any analysis requires decompression of the file (Loh, Baym and Berger, 2012).

Network traffic analysis is not the only area of scientific research where large volumes of data must be analyzed in an efficient manner. Genomic research is increasing datasets at a factor of nearly 10 times every year (Berger, Daniels, & Yu, 2016). Due to this, various new data compression techniques have been developed to compensate for storage constraints. Taking this one step further, in 2012, Loh, Baym and Berger proposed a compression algorithm that allowed them to perform genomic analysis directly on the compressed data, without having to be uncompressed (Loh, Baym and Berger, 2012; Loh, Baym and Berger, 2012a).

The purpose of this research was to incorporate the dynamic compression techniques of Loh, et al. (2012) into a new approach to reduce the size of the network data file storage, while maintaining data integrity to allow future reconstruction of events leading to an incident.

Problem Statement

There is no system that utilizes preprocessing of network traffic to remove duplicate and extraneous data prior to compression, decreasing the resultant file size to not only save storage space but to decrease the time required to compress and decompress. Network traffic audit trails are critical in cybersecurity, not just for the detection of current attacks, but also for the reconstruction of the events that led up to an attack, which can aid in mounting a defense against future similar attacks. In the HP example above, audit logs can be quite large, making them unwieldy for future analysis. There are many applications that can compress data files, but as the size of the original file increases, the time to compress and decompress the file also increases. Cybersecurity professionals require access to historic network traffic to reconstruct an attack. In large organizations generating network audit files, reconstruction can become an onerous task. Kalutarage, et al. (2015), was able to show promise in detecting protracted insider attacks at the network level, but due to the large amount of data required to detect activity over weeks and months, sampling techniques were employed. In doing so, only approximately

10% of network traffic was being analyzed, leaving 90% unanalyzed and potentially hiding stealthy insider activity, which is not a practical solution for forensic reconstruction. To be able to correlate activity across multiple packet flows and possibly multiple individuals requires more intensive analysis of internet traffic. Therefore, a method to reduce the volume of internet traffic while maintaining its integrity would be a valuable tool for cybersecurity analysists.

Dissertation Goal

The goal of this research was to develop a method whereby network traffic can be compressed in real time. It was previously discussed that there are many forms of audit logs. However, it is the larger packet capture logs required by network intrusion detection systems that is the focus of the research. As indicated by Kalutarage, et al. (2015), a malicious insider can perform their attack over weeks and months, necessitating the maintenance of audit logs over an extended period with its concomitant increase in file size.

Building upon prior research by Loh, et al. (2012, 2012a), a new compression algorithm was developed to take advantage of commonality in the data segments of network packets, such that a user's entire network traffic over an extended period could be captured for future analysis. The scope was limited to determining the viability of the compression technique; detecting malicious activity of any kind was outside the scope of this research. Unfortunately, no access to a network sufficiently large enough to generate a significantly large volume and variety of network traffic was available, so the genomic compression technique was tested using static network packet capture files. These files were generated from networks with additional traffic superimposed on top or from synthetic data created to represent network traffic. In all cases, the network capture files were continuous with no breaks in traffic outside of normal packet-to-packet separations.

Relevance and Significance

The feasibility of analyzing a large audit file is not the only issue facing organizations; the cost to maintain those large audit files can become substantial. For example, the published Google Cloud audit costs states one can store up to 50 GB of data access audit logs for 30 days at no additional charge. However, beyond 50 GB, there is an additional charge of $0.50 per GB per month, and if the logs need to be retained for a longer period, there is an additional charge of $0.01 per GB for the second month and $0.02 per GB for each month after that. Therefore, for the first petabyte (PB) of audit logs, there would be an additional charge of $475.00 and each PB beyond that of $500.00. If the logs need to be retained for one year, there would be an additional charge of $10 per PB for the second month, and $20 per PB per month for months three through 12, or $210 for the year. Although log files grow over time, if the average size of retained audit logs is 50PB, there could be a cost in excess of $400,000 per year.

The NIST Cybersecurity Framework provides an approach to managing cybersecurity risk; and is comprised of three parts: the Framework Core, the Framework Implementation Tiers, and the Framework Profiles (NIST CSFW, 2018). The Framework Core establishes five functional areas: Identity, Protect, Detect, Respond, and Recovery. The Detect functional area includes the use of network intrusion and protection devices, and audit event correlation and reporting; the Protect functional area includes use of audit logging; and the Respond functional area includes performing forensic analyses of events (NIST CSFW, 2018). The NIST Cybersecurity Framework is not the only guideline for cybersecurity practices, such as: the NIST Risk Management Framework (NIST SP 800-37R2, 2018); the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) publish the ISO/IEC 27001 Information Security Management Standard (ISO/IEC 27001, n.d.); and the Center for Internet Security provides the CIS Controls® (CIS Controls, n.d.)[2]. All these approaches to cybersecurity include the need for monitoring network traffic for malicious activity in real time and maintaining audit log files for future forensic analysis or reconstruction to determine how the malicious event happened and how to remediate the vulnerability. For example, Advanced Persistent Threats (APT) can last for weeks and months and the attacks contain multiple steps that individually are difficult to identify (Zou, Liu, Sun, and Singhal, 2020). In fact, three different APTs have been active for up to 20 years: Titan Rain since 2003, SkiPot since 2006, and Dep Panda since 2012. To determine the method and scope of an APT attack requires analyzing log files for the duration of that attack.

Thus, a method that can provide for better compression of network traffic which would allow for retaining more network traffic, could be beneficial to organizations. Not only by improving the ability to reconstruct possible malicious activity to implement cyber defenses against future attacks, but also by decreasing the financial impact to the organization for storage of those files.

[2] Although the listed security frameworks are utilized throughout both the private and public sectors, the author has had years of experience with the NIST Special Publications and its Risk Management Framework. Therefore, throughout this Dissertation Report, references to published cyber security guidance will be to NIST guidance only.

Barriers and Issues

The genomic compression technique is based on identification of common segments of data strings between a test segment and a dictionary of known segments (Loh, et al., 2012, 2012a). To take advantage of this technique with network traffic, the network traffic packets must contain substantial data segments. Although there are a number of protocols that are simple message protocols (request and reply messages with minimal or no data segment), it was a simple matter of identifying which parameters have data. No matter whether the Internet Protocol version 4 (IPv4) or the Internet Protocol version 6 (IPv6) packets are involved, the packet formats are standardized making the header sizes determinant. It therefore only required a simple calculation to determine if the packet contained any data segment. Those without data segments were not candidates for the compression technique and would be ignored during processing.

To handle packets with data properly required development of dedicated parsers. Each parser took advantage of the specific protocol format similar to the compressive techniques currently in genomic analysis – unique, critical data was maintained as is, while common, repetitive data was maintained offline once, and substituted with a link. The key issue was identifying what data must be available for analysis, and what can be compressed; what data could be totally ignored and possibly discarded; and what was the best file format for the minimized packets to allow forensic analysis.

The most difficult impediment to success was analyzing secure communications. Many organizations utilize secure communications, even within their Intranet – the more valuable the information, the more secure it will be. The best forensic analysis will be useless if the data segment of the packet cannot be decrypted and analyzed. For these approaches to be successful, they will require having access to the certificates/keys used to encrypt. However, having all requisite keys may be problematic. Another approach is called Transport Layer Security (TLS) break and inspect, also known as Transport Layer Security Inspection (TLSI). This is a process in which enterprise network traffic is decrypted, the decrypted content is inspected for threats, and then the traffic is re-encrypted before it enters or leaves the network (NSA, 2019). This process is not without issues and requires careful configuration and monitoring to ensure performance is not negatively impacted and the unencrypted data is not exposed, which could violate privacy laws, or worse, show usernames and passwords in plain text.

With the break and inspect system, audit logs covering a sufficient time span could be used to help reconstruct a general representation of the network traffic. This is not without challenge. If there are anomalous users in an organization, then their anomalous actions would appear in the audit logs. If you use the logs to group individuals and to emulate normal traffic, there might not be a clean base upon which you can detect anomalies – the existing anomalies would be considered normal. If access is being analyzed, then ignoring the encrypted data segment might not be a problem. For secure communications, the compression technique will not be as efficient, but should still decrease the overall storage requirement compared to normal compression techniques.

Assumptions, Limitations, and Delimitations

There are several issues that impacted the research. A major issue pertained to the number of transport protocols in the dataset that do not contain a data segment. If these packets were compressed as is without the benefits of genomic techniques, they would skew the compression ratio. So, an approach to analyze compression ratios ignoring uncompressible packets was required. Another limitation previously discussed is that secure communications without break and inspect would also skew the compression ratio since there will be fewer common payloads that can be saved once and substituted with a link. Since a component of the genomic compression technique was looking at commonality, mainly based on IP address and port combinations, it was assumed that there was no IP or Media Access Control (MAC) address spoofing, such that the identified sources and destination were actual, not spoofed. As to IP addresses, as will be discussed in the methodology section, not all IP addresses are static. Many organizations assign user IP addresses utilizing Dynamic Host Configuration Protocol (DHCP) which may assign a different IP address to a system the next time it connects to the network. Even without DHCP, users may have laptop computers and will connect from different locations or utilize secure Virtual Private Network (VPN) connections, also causing a different IP address to be assigned. So, without some method of definitively identifying a user each time they connect, long-term analysis of the compressed files may be disconnected and not accurately reflect users.

Summary

With the frequency of cyber threats, it has become important to be able to maintain network traffic log files for future reconstruction and forensic analysis. Unfortunately, with organizations generating upwards of hundreds of terabytes of network traffic per day, the sheer volume of traffic makes storage and analysis problematic. Although there are a variety of compression techniques available, they create static compressed files, that must be decompressed to be analyzed. In order to perform analysis of large audit log files, a method that can decrease the volume of data while maintaining data integrity for possible forensic analysis would be a valuable tool for cybersecurity.

Chapter 2

Review of the Literature

Overview

Cybersecurity professionals agree that it is a basic requirement to obtain audit trails (log files) to determine whether the actions taken by users could impact the security posture of the systems and enterprise and allow for reconstruction of those actions taken by the user leading up to the event (Marker, 2021; Shopp, 2020; NIST ITL Bulletin, 1997). A brief history of audit logging will be presented. Due to the physical size of audit logs and the need to maintain them for periods of time, common types of compression techniques will be provided. The review will cover general file compression algorithms, but will not be exhaustive, since the goal of this research is to apply the genomic compression techniques proposed by Po-Ru Loh, Michael Baym, and Bonnie Berger in 2012. Their approach entailed identifying data in a test string, that is common with a known string, and replacing it with links to that known data, thus reducing the size of the test string. Therefore, the review of compression algorithms is to identify the pros and cons of the different algorithms and propose possible solutions that could be utilized in a “production” implementation. To better understand the value of the approach by Loh et al. (2012), it is important to understand the various search and approximate match algorithms that have been used in genomic research. A brief review of those techniques will be provided, including algorithms used in such applications as spell checkers and identifying the longest common substring – these are instrumental to this research.

Context

In 1979, David Hanson identified three different classes of computer vulnerabilities: threats to the physical system; individuals who threaten the integrity of or loss to the system from the outside; and individuals who threaten the integrity of or loss to the system from the inside (Hanson, 1979). However, it was not until the high-profile cases of Robert Hanssen, Chelsea (a.k.a. Bradley) Manning, and finally Edward Snowden came to light, that the issue received the attention it deserved. On February 20, 2001, the FBI announced the arrest of Robert Hanssen for selling highly classified national security information to the Russians ("Robert Hanssen," 2001). Between November 2009 and May 2010, Private First Class Bradley Manning passed on hundreds of thousands of United States Government classified documents to WikiLeaks, who released them between February 2010 and the autumn of 2011 ("Chelsea Manning," n.d.). On October 7, 2011, President Barack Obama signed Executive Order 13587, the "National Insider Threat Policy" (Obama, 2011). However, that was not a deterrent. In May 2013, Edward Snowden, an NSA analyst working for contractor Booz Allen Hamilton, leaked the details of several top-secret surveillance programs to the media ("Edward Snowden," n.d.). Although the Hanssen case pertained to hard copy documents while the Manning and Snowden cases involved release of computer (or computerized) documents, all three instances fall into the category of exfiltration, a term meaning the unauthorized removal or transfer of data. There is a plethora of references to research on exfiltration, and although exfiltration might be the end goal of an insider, exfiltration itself is not within the scope of the proposed research. However, what these three individuals have in common with the proposed research is that they were trusted, authenticated users (although Hanssen did not commit a computer crime, he was authorized to access the documents that he stole). The threat posed by a trusted insider can be much more pervasive than exfiltration, it can extend to changing data, damaging systems, and committing fraud.

Audit Log Files

Audit trails (logs) can assist an organization in detecting security violations (NIST ITL Bulletin, 1997). Since audit logs are created at the time an event is occurring, that event might not be stopped. However, reviewing audit logs on a routine basis and taking proper actions could have decreased the magnitude of the security leaks by Manning or Snowden.

Modern audit logging (audit trails) really did not begin until the introduction of the UNIVAC in 1954 and the first computer accounting systems (Singleton, 1996). By the 1980's, classical "paper trails" were vanishing in favor of computerized systems (Anderson, 1981). Anderson goes on to discuss how electronic audit trails can be useful in detecting possible crime that normal financial auditing did not. By the 1990's, the use of audit logs had become common place, leading to the NIST Information Technology Laboratory (ITL) issuing the ITL Bulletin 1997-03, "Audit Trails" that formally addresses the uses and advantages of audit logs, including their use in intrusion detection (NIST ITL Bulletin, 1997).

Under the Federal Information Security Management Act (FISMA) of 2002, Public Law 107-347, NIST was tasked with providing guidance for the security certification and accreditation of information systems of the U.S. federal government. This culminated in the issuance of the NIST Special Publication (SP) 800-37, "Guide for the Security Certification and Accreditation of Federal Information Systems, and a multitude of supporting special publications. One of those is NIST SP 800-53, "Recommended Security Controls for Federal Information Systems," with a final release in February 2005. The security control catalog is organized into three general classes of security controls (management, operational, and technical) and 17 different families. The Audit and Accountability family is considered a technical control and consists of 11 base security controls, with an additional nine (9) security control enhancements spread across the base controls, for a total of 20 Audit and Accountability security controls (NIST SP 800-53, 2005).

The NIST guidance has gone through several revisions over the years, such that currently, NIST SP 800-37 is at Revision 2, "Risk Management Framework for Information Systems and Organizations, A System Life Cycle Approach for Security and Privacy," released in December 2018, and NIST SP 800-53, Revision 5, "Security and Privacy Controls for Information Systems and Organizations," released in September 2020. NIST SP 800-53, Revision 5 has expanded to 20 control families which now include security controls aimed at protecting privacy. The Audit and Accountability (AU) family now consists of 16 base controls and 40 control enhancements, for a total of 56 security and privacy controls devoted to auditing and accountability (see Appendix A, NIST SP 800-53r5, 2020). Without discussing every security and privacy control, there are several specific controls that are pertinent to this research: 1) AU-3 "Content of Audit Records" provides guidance on the type of events and information that is to be maintained in an audit record; 2) AU-4 "Audit Log Storage Capacity" dictates that systems must comply with an organizations audit storage capacity, and the requirements for backing up audit log files as they reach their configured maximum size; 3) AU-6 "Audit Record Review, Analysis, and Reporting" is critical to maintaining a secure system by providing guidance on the type and frequency of analysis of audit logs; 4) AU-7 "Audit Record Reduction and Report Generation" supports the goal of this research, in that the organization must "a. Supports on-demand audit record review, analysis, and reporting requirements and after-the-fact investigations of incidents; and b. Does not alter the original content or time ordering of audit records" (NIST SP 800-53r5, 2020, p 72); 5) AU-11 "Audit Record Retention" provides guidance on the length of time audit logs must be maintained; and 6) AU-13 "Monitoring for Information Disclosure" provides guidance on auditing for unauthorized use of information, including exfiltration, making unauthorized copies of information, and monitoring for evidence of unauthorized disclosure of information.

One related security control to AU-13 is, SI-4 "System Monitoring," Enhancement 2 "Automated Tools and Mechanisms for Real-Time Analysis" which dictates the use of automated tools and mechanisms to support analysis of possible security events in near real time. This can entail the use of SIEM technologies (NIST SP 800-53r5, 2020, p 365). In 2005, the Gartner Group discussed the combination of two systems – Security Information Management (SIM) and Security Event Management (SEM) – into a single tool, the SEIM (Buecker et al., 2010). Thus, the SIEM combines the security information (risk), event, and threat data into a single system that can correlate this varied information, identifies possible deviations, provides reporting capability of results, and provide for detection and remediation of possible security issues (IRS, n.d.; Mokalled et al., 2019).

Log files take many forms. The NIST SP 800-92 Guide to Computer Security Log Management provides a very broad listing of types of audit logs and events to capture (NIST SP 800-92, 2006). Many organizations have developed their own policies on event logging, such as the University of California Berkeley. They have published their “Security Audit Logging Guideline” that provides detailed guidance on the information that must be included in audit logs for Operating System (OS) Events, OS Audit Records, Application Account Information, and Application operations. Although they allow for variance in the content of the different audit events, at a minimum, each record must contain the “timestamp, event, status, and/or error codes, service/command/application name, user or system account associated with an event, and device used (e.g., source and destination IPs, terminal session ID, web browser, etc.)” (UC Berkeley, n.d.).

All federal information systems must comply with NIST guidance, and therefore must comply with the audit file creation, retention, and analysis guidelines provided by the security and privacy controls (FISMA, 2002). In accordance with Title 44 U.S. Code, Sec. 3554, all components of nonfederal systems that process, store, or transmit Controlled Unclassified Information (CUI), or that provide security protection for such components are subject to the same requirements as federal system as provided in NIST SP 800-171 Revision 2. This NIST guidance further states that these nonfederal systems must follow the guidance provided in NIST SP 800-53 (NIST SP 800-171r2, 2017). Thus, all affected nonfederal systems must comply with all the auditing requirements of federal systems.

Intrusion Detection

Intrusion detection is the process of monitoring network communications for unauthorized accesses or activity, logging the findings, and reporting those findings (Liao et al., 2013; Scarfone & Mell, 2007). An Intrusion Detection System (IDS) monitors for possible unauthorized activity – passively, offline – and provides notifications of possible activity for further processing (OWASP, n.d.; Scarfone & Mell, 2007). Intrusion prevention not only performs intrusion detection, but takes that one step further, by actively attempting to stop the attack in real time (Liao et al., 2013; OWASP, n.d.; Scarfone & Mell, 2007). Although prevention of intrusions is a very important tool for cybersecurity professionals, only intrusion detection can require storing large amounts of data to detect long-term threats such as APTs. Although the actual detection of intrusion is not within the scope of this research, providing a mechanism to collect and maintain historical network traffic would assist cybersecurity analysts.

Some IDSs can store all data relating to sessions for short periods of time to detect a possible attack and the steps taken during that attack (Kent et al., 2006). Other systems perform deep packet inspection in which detection is based on specific data or payloads. This is usually performed at the application layer for a specific list of protocols (Awati & Scarpati, n.d.). Regardless of the category of intrusion detection or the underlying technology used to detect, many sensors incorporate packet capture to store many packets for off-line analysis (Scarfone & Mell, 2007). These systems could benefit from the research provided here.

Signature-based detection utilizes a known list of signatures and compares new activity against the list looking for matches (OWASP, n.d., Scarfone & Mell, 2007; Tek-Tools, 2020). Signature-based detection is considered the simplest since it only requires comparison to the known signature database. Unfortunately, signature-based detection is ineffective at identifying new threats without a known signature and are also generally ineffective in identifying threats of many application protocols (Scarfone & Mell, 2007). Although signature-base intrusion detection is normally applied in real time, should a new attack vector be discovered with new signatures, analyzing historical network traffic for these signatures could identify previously unknown attacks against a system.

Anomaly-based intrusion detection (AID) covers a broad range of approaches to detect network activity that differs from the norm; from the expected. Much like signature-based detection, AID requires the knowledge of what is considered a normal network profile and compares the current profile to the known profiles to identify anomalous activity (Scarfone & Mell, 2007). However, there are newer approaches to anomaly-based detection that are more-effective in identifying previously unknown behavior. Punithavathani, Sujatha, and Jain (2015) present a multi-step approach to insider threat detection. The first stage is to monitor network activity and look for known patterns of anomalous activity. The second stage involves reviewing historic network traffic (log files) to identify previously undetected user anomalies. If a new anomaly is identified from the log files, the associated network traffic is reviewed to update the pattern recognition of stage one, thus improving the chance of future detection. This second stage of the Punithavathani, Sujatha, and Jain (2015) approach could benefit from the research presented here by providing smaller files for analysis.

Network IP Flow Techniques

A packet flow consists of the total sequence of network packets between a source computer and its destination and is specific to not only the IP addresses of both devices, but also the specific protocol and ports utilized (Claise, Trammell, & Aitken, 2013; Kerner, 2021; Petryschuck, 2019). Several researchers have utilized network IP flow-based analysis for intrusion detection as well as detecting insider threat activity. The origin of network IP flows goes back to the Internet Accounting Working Group (IAWG) of the Internet Engineering Task Force (IETF) in 1991. The intent of the IAWG was to develop an efficient method of monitoring network traffic for the purposes of accurate billing for Internet use as well as a method of maintaining access information that may be required during an investigation of a serious crime (Mills, Hirsh, & Ruth, 1991).

As network speeds and traffic volume increase, deep-packet examination for intrusion detection becomes unwieldy. One answer has been the use of network flow monitoring; a method by which the connection and possible intent behind the connection can be analyzed. By 2010, intrusion detection using IP network flow had become an established area of research. Sperotto et al. (2010) provided an overview of the pertinent research. However, none of the presented systems could perform analysis of traffic flow over extended periods of time (Sperotto et al., 2010). Use of network flows continues today. In 2017, Alaidaros and Mahmuddin presented an approach where they modified the open-source Bro IDS[3] to make intrusion detection closer to real-time. However, this approach ignores the payloads altogether, and cannot detect possible intrusions based on content of the payload, nor detect what data is being transferred (Alaidaros and

[3] Bro began in 1995 at the Lawrence Berkely National Laboratory, and was rebranded as “zeek” in 2017. The product can be obtained from either https://bro.org or https://zeek.org.

Mahmuddin, 2017). According to Hofstede et al. (2014), the size of flow repositories can exceed tens of terabytes, so network flows alone cannot solve the data volume problem, unless a method is available to decrease the size of the flow repositories.

Compression Techniques

Compression is the process by which the quantity of content data is reduced without excessively impacting the quality of the original content. The end goal is to minimize the number of bits in digital media required for storage and/or transmission, in an efficient, cost-effective manner (Kavitha, 2016). In 1981, Hassan K. Reghbati divided compression into two categories based on the results of decompression. Nonreversible compression occurs where data deemed insignificant is removed, while relevant data is maintained. Reversible compression occurs when all the data is considered relevant, such that decompression returns an identical copy of the original data (Reghbati, 1981). Today, nonreversible and reversible have been replaced with the terms lossless and lossy[4].

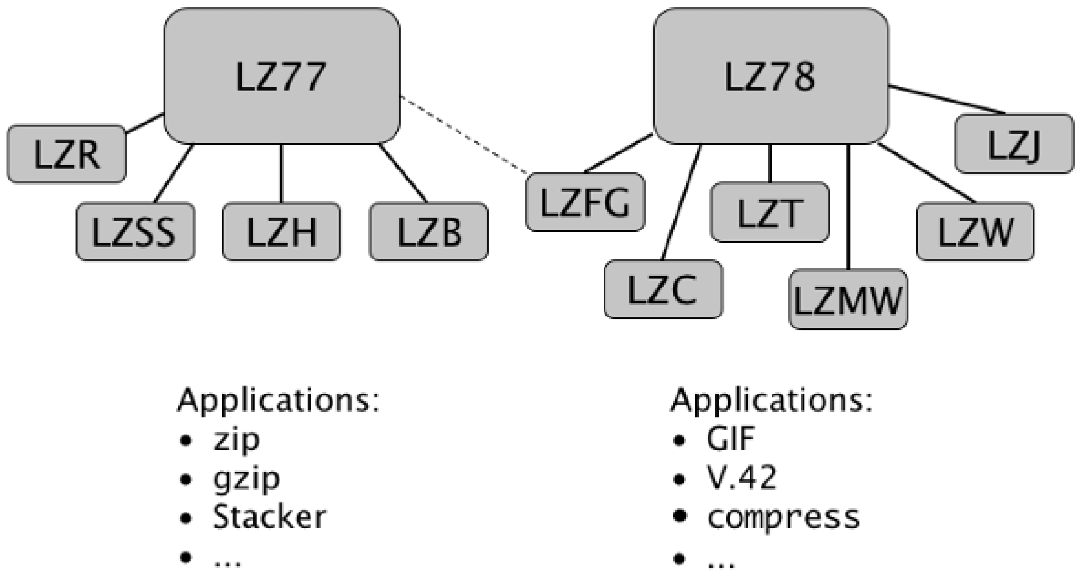
Mahdi, Mohammed, and Mohamed (2012) discuss the two general categories of compression, lossless and lossy. Lossless compression identifies and eliminates statistically redundant bits. This allows for recovery of an exact copy of the original upon decompression. Lossy compression identifies marginal information and eliminates that during compression. However, the exact original content cannot be recovered upon decompression (Mahdi, Mohammed, and Mohamed, 2012).

[4] Although an extensive literature search was performed, the origin of the use of the terms lossy and lossless was not determined. Various articles in the 1990-1991 timeframe refer to these terms, but do not state an origin, and there were no references found prior to 1990.

Lossless Compression

Many lossless algorithms have two separate phases: in the initial phase, a statistical model of the input data is created, identifying codewords; and in the final phase, the input data is mapped to the codewords creating the smaller output file (Fitriya, Purboyo, & Prasasti, 2017; Kavitha, 2016). In his book, A New Kind of Science, Stephen Wolfram (2002) states that the Morse code (invented in 1838) is a form of lossless compression, since it substitutes shorter code sequences for the most-common letters of the English language. In 1952, David Huffman presented a compression method that used codewords based on probabilistic analysis of the data, with the more-common characters receiving the shortest codewords (Huffman, 1952). His approach has subsequently been known as Huffman encoding.

Figure 1 - Lempel-Ziv Algorithm Family (Source: Zeeh, 2003).



In 1977 and 1978, Jacob Ziv and Abraham Lempel proposed two lossless compression algorithms (LZ77 and LZ78), that were the beginning of a whole family of compression algorithms as depicted in Figure 1 (Zeeh, 2003). In 1984, Terry Welch published a modification (LZW), that has become possibly more popular than the original Lempel-Ziv (LV) algorithms (Zeeh, 2003). One change made by Welch pertains to the codewords (called translation table and string table, interchangeably by Welch). LZW utilizes 12-bit codes, so the table is fixed at a maximum of 4096 entries, whereas LZ77 and LZ78 are not limited to table size. Another is pre-filling the table with all possible characters in the input string, and when a match is not found, it is added to the table (Welch, 1984). The most-known implementations of the LZW algorithm are in the Graphics Interchange Format (GIF) for image compression, the UNIX compress command, and in the original PKZIP archive compression tool (Hosseini, 2012; Wolfram, 2003). Other lossless compression algorithms include: 1) run-length encoding (RLE) in which runs of common characters (bits) are stored in two parts, one for the symbol and one for the count (Capon, 1959). This is mainly used in FAX machine transmission due to its efficiency on black and white text documents (Hosseini, 2012); 2) Portable Network Graphics (PNG) was developed in 1996 as a replacement to GIF, which required license fees. In addition, it could support up to 24-bit color graphics (Aguilera, 2006; Hosseini, 2012); and 3) Tagged Image File Format (TIFF) used to store images, including photographs and is the most-popular raster file format. It is commonly used as the method to store FAX rasterized output (Aguilera, 2006).

Lossy Compression

As stated above, lossy compression is utilized where there is data deemed not relevant to the original or is less significant and therefore can be discarded without seriously impacting the quality of the data upon decompression (Fitriya, Purboyo, & Prasasti, 2017; Kavitha, 2016; Mahdi, Mohammed, & Mohamed, 2018). In compression review articles, lossy has been associated with image, audio, and video file storage, as well as with streaming transmissions (Kavitha, 2016; Smith, 2010). In discussing lossy compression, Kavitha limited the applicable protocols to JPEG.

Joseph Fourier identified that in many cases, you could decompose functions into sums of sine waves and frequencies. With an audio file, you can then drop the lowest and highest frequencies that are outside the range of human hearing and save significant space, without impacting the quality of the final audio output (Hosseini, 2012; Kavitha, 2016; Smith, 2010).

Fractals are geometric structures that appear similar at different scales. Fractal compression relies on that feature and identifies segments of an image that are like other segments, converting them into “fractal codes” which are used in reconstruction of the encoded image (Smith, 2010). Having identified common, similar segments, fractal compression needs only address the differences, thus obtaining significant compression ratios. This makes fractal compression valuable for transferring large amounts of data, in particular streaming services, such as NetFlix (Smith, 2010).

Lossy compression techniques usually fall within three areas: transform coding (TC), discrete cosine transform (DCT), and discrete wavelet transform (DWT) (Kavitha, 2016). TC works best with audio signals or images and requires knowledge of the application to determine what data can be removed. Although the result does not match the original, the difference is insignificant. The result of a DCT is a sum of the cosine functions at different frequencies. This approach is like Fourier transforms, except cosine waves are utilized, as opposed to sine waves. Unlike DCT, DWT does not resemble

Fourier transforms, in that the signal is decomposed into a set of orthogonal wavelet base functions (Kavitha, 2016).

The two main lossy compression algorithms are the Joint Photographic Experts Group (JPEG) and the Moving Pictures Expert Group (MPEG)[5] (Hosseini, 2012). JPEG is aimed toward compressing images that can either be greyscale or up to 24-bit color. JPEG offers several options, including adjusting the compression rate, the luminance (brightness), the color saturation, and the hue Aguilera (2006). The Moving Pictures Expert Group (MPEG) compression entails transforming a stream of discrete samples (frames of a movie) into a bit stream of tokens. Using the fractal compression discussed above, from frame to frame, only the differences are coded, along with any information for any moving parts (Hosseini, 2012). Depending upon the application, the difference between loss of audio/video quality will be based on the transmission speed or storage size requirements. A modern compact disc (CD) can contain a maximum of 640 MB of data (Mahdi, Mohammed, and Mohamed, 2012). This can hold only about one hour of uncompressed high-fidelity music, two hours of compressed lossless music, up to seven hours of music in the MP3 format, or approximately 200 hours of a voice recording.

Compressed Bitmap Indexes

Although compressed bitmaps are technically lossless compression and are integral to the underlying algorithms utilized in the various lossless compression techniques discussed above, when utilized as indexes, they have a very different purpose than previously discussed. Various commercial relational database management systems

[5] MP3 is a shortened version of MPEG-1 Audio Layer 3, so any reference to MPEG includes MP3. MPEG4 (MPEG-4) is a video decoding algorithm, a video codec. However, unlike MP3, MP4 is not an “abbreviation” of any MPEG codec. Rather, MP4 is a media container format and that allows storing still images, subtitles, video, audio, and other essential data (Dolina, 2021).

(RDBMS) utilize bitmap indices in read-only views in data warehouses, since they are more efficient than other methods, such as binary trees. However, updates have been shown to be more process intensive (Canahuate, Gibas, and Ferhatosmanoglu, 2007; Chen et al., 2015).

The Byte-aligned Bitmap Compression method (BBC) abandons using run-length encoded compression since it requires decompression to perform many database functions. Rather, BBC is a byte-aligned, byte-sized bitmap that allows operations on the compressed bitmap after a simple merge routine that only requires partial decompression (Antoshenkov, 1995). BBC is very efficient, but only on data with low cardinality; it is less efficient than uncompressed bitmaps for data with higher cardinality.

The Word-Aligned Hybrid (WAH) code provides for an improvement in logical speed on the order of a magnitude over BBC. However, being word-aligned versus byte-aligned, there is a small increase in the space (memory) required by WAH. Another improvement over BBC is the ability to be applied to data with high cardinality (Wu, Otoo, & Shoshani, 2002; Wu, Otoo, & Shoshani, 2006).

Update Conscious Bitmap (UCB) Indices can update the relevant bitmaps with minimal increase in time. With previous methods, updating required access to all bitmaps in the system and modification to those that are relevant. UCB only requires modification of a single bitmap per indexed attribute, a significant savings (Canahuate, Gibas, & Ferhatosmanoglu, 2007).

Run-Length Huffman (RLH) compression provides good query times while maintaining compressed bitmaps. Instead of normal run-length encoding utilized by BBC and WAH, the RHL compression algorithm utilizes a version of Huffman encoding, in which symbols are replace with bit strings – the more-frequently occurring symbols are associated with shorter strings (Stabno & Wrembel, 2009).

The Position List Word Aligned Hybrid (PLWAH) improves upon WAH by repurposing bits that are never used by WAH to hold information for the set/unset bits of a 0/1 run. This way, the compressed bits are half the size of WAH, lending to faster processing (Deliège & Pedersen, 2010). Lemire, Kaser, and Aouiche (2010) presented the Enhanced Word-Aligned Hybrid (EWAH) which 99.9% of the time, will not generate a bitmap that is larger than the original, uncompressed bitmap. Another change is that EWAH utilizes two types of words used to store the number of clean words and the number of dirty words (Lemire, Kaser, & Aouiche, 2010). Colantonio and Di Pietro (2010) present the *Co*mpound '*n*' *C*omposable *I*nteger *Se*t (CONCISE) compressed bitmap index. When there are few bits set followed by a long sequence of unset bits, the WAH compression approach was improved and a decrease in the size of the compressed bitmap was achieved, sometimes up to 50% less (Colantonio and Di Pietro, 2010). Much like with EWAH, the decrease compressed bitmap size led to performance improvement, in addition to the space savings.

There are other variants of the WAH bitmap compression algorithm: the Partitioned Word-Aligned Hybrid (PWAH) compression that utilizes compressed bit vectors (van Schaik & de Moor, 2011); another variation introduces a new block type containing "draggled fills" that can address dirty literals inside a fill. Unlike the other two fixed-length blocks with WAH, the draggled fill block is variable length, based upon the number of dirty literals it contains (Schmidt, Kimmig, & Beine, 2011). The Variable Aligned Length (VAL) WAH (VAL-WAH) allows for setting the segment length prior to running, as opposed to WAH that fixes the length to the size of CPU architecture word, minus 1 – a 32-bit system would utilize 31-bit lengths, while a 64-bit CPU would utilize 63-bit segments. Depending upon the segment length, the encoding compression scheme can change. In this way, the compression can be optimized by the user (Guzun, Canahuate, Chiu, & Sawin, 2014). The one downside of this approach is that it requires operator interaction to configure the system at run time, implying it might not be practical as a background process.

The Roaring bitmap compression scheme partitions the space into chunks and stores dense and sparse chunks differently (Chambi, Lemire, Kaser, & Godin, 2016). 32-bit indexes are broken into chunks of 2^{16} integers and separated into two different containers – one for identical 16 most-significant bits and a different container to store the 16 least significant bits. The containers are processed based on the container density. If there are no more than 4096 16-bit integers in the container, a sorted array of packed 16-bit integers is utilized, otherwise, the container is stored using a 2^{16}-bit bitmap (Chambi, et al., 2016).

Wu et al. (2016) presented a bitmap compression scheme in which the raw bitmap indexes are divided into many bitmap snippets. Common affixes are then merged among the various bitmap snippets. This approach is aptly named the Common Affix Merging with Partition (CAMP) (Wu et al., 2016). Testing indicates that CAMP can create smaller indexes when compared to Roaring, WAH, CONCISE, and the *COMP*ressed *A*daptive inde*X* (COMPAX), although when density is greater than approximately 6%, Roaring exhibits comparable storage requirements (Wu et al., 2016).

In 2016, Wen et al. introduced the *MA*ximized *S*tride with *C*arrier (MASC). The basis for MASC is the implementation of a new bitmap index coding scheme that would exhibit improved compression ratios. Unlike PLWAH and COMPAX, MASC is not limited to 31-bit runs. Rather, runs in MASC are as long as possible. Testing against the CAIDA dataset[6] indicate that MASC provides improved compression ratios of approximately 10% (Wen et al., 2016).

The *Co*mpressing *Di*rty *S*nippet (CODIS) compression scheme was presented by Zheng, Liu, Chen, and Cao (2017). It is based on WAH, but fewer bits are used to represent the bit string while maintaining inter-bitmap operation efficiency. Using the CAIDA 2016 dataset, comparisons were made to WAH, PLWAH, and COMPAX as to file size, and four different performance times: encoding, decoding, unions, and intersections. CODIS was the best in only decoding time. COMPAX required the smallest file size, and WAH performed the best on the remaining three performance evaluations (Zheng et al., 2017).

Network Compression

There have been attempts at compressing network traffic, with varying degrees of success. In 2000, Spring and Wetherall presented a protocol-independent technique to identify and eliminate redundant data from web content. However, their experimentation was as a user-level process running on a PC and analyzing a static file, so there is no indication whether this technique would truly be applicable in the enterprise to decrease network traffic (Spring & Wetheral, 2000).

[6] The Center for Applied Internet Data Analysis (CAIDA) is based at the Supercomputer Center at the University of California San Diego. CAIDA currently supports a large number of datasets that can be obtained from https://catalog.caida.org/search?query=types=dataset%20links=tag:caida%20.

Fusco, Stoecklin, and Vlachos (2010) presented COMPAX format that is designed to perform on-the-fly compressing, archiving, and indexing of streaming network data. There are two components in the process, an archiving backend to compress the incoming data, and COMPAX, the compressed bitmap index. Their key to decreasing the size of the compressed bitmap index is to use a codebook of just four word types that are representative of streaming data (Fusco, Stoecklin, & Vlachos, 2010).

Kyriakopoulos and Parish (2010) describe an approach to compress time-based monitoring and measuring metrics from high-speed network traffic, using wavelets. The approach is considered lossy compression since some information is lost in the process. Use of wavelets is not truly a compression technique. Rather, it is a method that transforms the data into a different view of the data (Kyriakopoulos & Parish, 2010). Wavelet compression shows a significant improvement in compression compared to Gzip and Bzip2, but the approach was proposed for network monitoring and performance measurements, not general network traffic (packet capture) compression.

Sardari, Beirami, Zou, and Fekri (2013) proposed a compression scheme to decrease network traffic. Their approach was to utilize a form of memory-assisted compression, where compression is performed based on previously seen sequences residing in memory. To accomplish this, a content-aware clustering algorithm was developed that grouped packets in memory that can be compressed. Sardari et al. (2013) relied on individual nodes in a network learning the data statistics in network traffic to aid in compression. There are possible problems with this approach. The content-aware clustering uses the Hellinger distance metric. However, the authors state the algorithm is too complex for real-time use and needs to be performed offline. They also tested using captured packets from content servers. These two issues indicate that the concept is not practical for large enterprise networks where there is a wide variety of high-volume network traffic.

Wen et al., (2014) proposed a new bitmap index encoding algorithm to compress network traffic for later analysis. SECOMPAX (Scope-Extended Compressed Adaptive Index) is claimed to perform better compression ratios and faster encoding schemes than other bitmap index compression algorithms. They performed experiments on an Internet traffic dataset obtained from CAIDA. The results indicated improved compression rates over other algorithms, although the encoding time took longer. Regardless, the Wen et al., (2014) concept is another form of data compression that requires decompression of the data for analysis, and not directly applicable to real-time analytical work. However, for the proposed dissertation research, SECOMPAX might prove valuable for compressing the known, repeatable sequences in the network traffic.

Genomic Data Compression

There has been a limited attempt to extend genomic compression techniques to network traffic. Oehmen, Peterson, and Dowson (2010) presented a paper to equate cyber-attacks to the evolutionary concept of survival of the fittest. The most effective are reused and even improved by future generations. Historically, this has made detection difficult – relying on a signature that now does not match the actual attack. To counter this, the authors present a detection approach that utilizes genomic algorithms that identify similar, although not exact strings. Thus, the authors claim to be able to identify evolved, perhaps obfuscated attacks from their similarity to a previous attack, regardless of mismatches, insertions, or deletions.

To evaluate this approach, Oehmen et al. (2010) utilized warning messages from Windows Active Directory (AD) log files. Each message was mapped to a unique character and a string of these characters was created from the messages over one day from 508 servers. The average length of a text string was 10,561 characters; the genomic analog would be 508 proteins with an average of 10,561 genes. The results were promising in being able to show similarities between cyber entities without prior knowledge. However, being based on warning messages from Windows AD log files, limits the potential use to Windows networks and only network activity involving AD.

Loh, Baym, and Berger (2012) present a compression algorithm that takes advantage of the redundancy that occurs in gene sequences. They replace the redundant segments with links to those segments, such that only the pertinent differences are left intact. Should they need access to the redundant portion during analysis, the link provides ready access. Their justification for this concept is to reduce the extremely large volume of genomic data that must be analyzed at any given time to a more manageable volume. Loh et al., (2012) state that genomic data is doubling every four months, while according to Moore’s Law, processing power and storage capacity are only doubling every 18 months, so some analytical method that can offset this disparity is critical. Although there exist numerous compressive algorithms utilized in genomic research, they are solely for saving storage space – the data must be uncompressed to be analyzed.

Loh, Baym, and Berger (2012a) provided more detail in supplemental material, which is available from the journal website. Although this compression technique is applied to a genomic database, it is applied sequentially as though there is an incoming stream of DNA, similar to network traffic. During the read phase, the system builds the main data structures: the unique database and the link pointers. The main issue is the identification of redundancy. For this, they utilized a variation of BLAT (BLAST-like alignment tool; BLAST: Basic Local Alignment Search Tool) to identify an acceptable alignment between a sequence fragment and an existing reference in the database. That portion is then replaced with a link to the reference with an edit script of the differences. With only a small percentage of genes being different, this approach greatly decreases the size of the database, while maintaining the ability to analyze directly.

In 2012, Oehmen, Peterson, and Teuton expanded upon the Oehman, Peterson, and Dowson (2010) paper. As in their earlier paper, the evolutionary aspects of cyber-attacks are discussed and the researchers state that cyber attackers change their approaches “more rapidly than our ability to recognize them” (Oehmen et al., 2012). On the surface, that is a true statement, but modern intrusion detection systems have combined signature and anomaly detection modules. So, although the signature may change, the anomaly detection module could very well detect the attack. However, if the new attack is an obfuscated version of a previous attack, the adoption of biological modeling techniques might be of benefit. Not only should their approach be able to identify a new attack but could possibly determine the type of attack based on its family.

A misleading statement made in the Oehmen et al. (2012) paper is that attackers can probe detection systems to determine what types of attacks can be identified. That is not totally accurate – a critical cybersecurity control is to ensure that discovery is turned off on network devices and default usernames and passwords are changed so that attackers cannot log onto these devices either (NIST SP 800-53r5, 2020). Thus, a probe would not yield any important information. In fact, by probing the IDS, that action itself could trigger an anomaly event. So, the premise of Oehmen et al. (2012) for recommending that detection systems become moving targets is not particularly warranted.

Regardless, the approach by Oehmen et al. (2012) of using genomic algorithms for evolutionary gene sequence drift to detect similarities in network traffic, software execution, or event files is promising. However, what is not clear is the real-time capability. The approach involves analysis of audit logs or software code, neither of which is particularly read as a time series. This approach is thus more closely representative of the underlying genomic algorithm that is designed to work on gene sequence databases. As with the approach by Teuton, Peterson, Nordwall, Akyol, and Oehmen (2013), Oehmen et al. (2012) are forcing the cyber events being analyzed to fit a gnomic model. In this case, similar events are mapped to a single character and then ScalaBLAST is used to identify families. With a limited alphabet required by ScalaBLAST, the scope of detection capabilities could be restricted. Regardless, the possibility of identifying similar, malicious sequences that do not match a known signature is a valuable tool for cybersecurity.

Teuton, Peterson, Nordwall, Akyol, and Oehmen (2013) presented an approach to detect malicious actions, by applying compression techniques developed for analyzing extremely large gene sequencing datasets. The approach allowed them to discover traffic signatures dynamically and detect anomalies without resorting to expert-defined signatures. Their system, called LINEBACkER (they do not define that acronym, in this paper nor any of their earlier published papers), utilizes a variation of BLAST (Basic

Local Alignment Search Tool) to discover patterns in network traffic: ScalaBLAST is the standard BLAST program recompiled to utilize multiprocessing.

One justification for selecting genomic techniques, is that antivirus models require having signatures of known malicious actions, which may or may not match the current patterns. Standard endpoint security programs are only effective on their host and cannot be applied to general network traffic. However, gene searching algorithms look for related sequences, not exact matches (Teuton et al., 2013). Thus, they can find attacks that are designed to defeat a signature-based system through obfuscations – insertions of extra commands or pieces of superfluous data.

The implementation displayed promising results. However, Teuton et al. (2013) forced network traffic to conform to the FASTA file format, used in genomic research. The problem with that is the "alphabet" is extremely limited, in this case, to 14 characters. To accomplish this, they mapped pieces of the packet using a custom mapping table, which was not provided. The results from ScalaBLAST are then clustered using a greedy hierarchal nearest neighbor algorithm to identify groups, or signatures of network traffic sequences. The authors state that these dynamic signatures can be used to identify traffic that can be allowed or should be blocked and is easier to maintain than either black- or whitelists. However, since the approach by Teuton et al. (2013) is based on URL and content, it cannot distinguish between authorized and unauthorized users, only where the URL or content is allowed or not.

Approximate Matching

A key component of the Loh et al. (2012) work is the performance of approximate matching of gene sequences. However, as has been discussed, mimicking this class of matching algorithms, such as presented by Needleman and Wunsch (1970) and Smith and Waterman (1981), would be too restrictive being designed to work on a DNA sequence, which is made up of stings of 20 different amino acids, that are comprised of three nucleotide pairs (Smith, 2008). Rather, network traffic is comprised of an unsigned character string, that contains 256 possible values, counting zero.

In 1950, Richard Hamming presented a method to determine the number of changes required to turn one string into another string, on a character-by-character basis (Hamming, 1950). This was a simple algorithm but was only practical for strings of identical length and was based solely on position in the string (Korstanje, 2020).

In 1965, Vladimir I. Levenshtein introduced an algorithm that identifies the difference between two strings by determining the minimum number of single-character insertions, deletions, or substitutions required to change one string into the other. The calculated difference is referred to as the edit distance and was an indication of the number of changes necessary to transform one string into another string (Levenshtein, 1966). In literature and in usage, edit distance is now referred to as the Levenshtein distance and is the total number of insertions, deletions, or substitutions required (Wu, 2021). A variation on the Levenshtein distance is the Damerau-Levenshtein distance, which include single character transpositions to occur – adjacent characters are reversed in performance of the transformation.[7]

There are two main advantages to the Levenshtein distance over the Hamming distance: 1) the Levenshtein distance is not position specific, so the matching portions

[7] Although this variation is attributed to Damerau, an exhaustive literature search has not identified any publications by Damerau on this variation, nor any publications that attribute this to him. Regardless, the inclusion of transpositions into the Levenshtein distance is associated with him.

can occur anywhere in either string; and 2) the Levenshtein distance works on strings of different lengths, compared to the equal-length string restriction of Hamming (Korstanje, 2020). Although both these algorithms are relatively simple, they rely on maintaining a 2-dimensional array where the first string is across the x-axis and the second string down the y-axis. Although this is not too memory-intensive for strings of reasonable length, it is not the most-efficient approach for longer strings (KokiYmgch, 2018).

Hirschberg (1975) solved the space issue with a new approximate matching algorithm based on the Divide and Conquer algorithm. The large matrix is divided into smaller matrices where there is a single position used to pass between them. As the algorithm finds the smallest value in the two matrices, another piece of the larger matrix is added and calculation resumes. This subdividing continues until the entire range of the matrix has been covered, yielding the result (Hirschberg, 1975; KokiYmgch, 2018). Although this algorithm can thus more-easily handle larger matrices (longer strings), it is much more complex and requires recursion that could lead to stack overflow on very large strings, so may not be a candidate for this research.

Summary

Cybersecurity professionals agree that obtaining and maintaining audit files (logs) is a critical effort in combating cyber threats (Marker, 2021; Shopp, 2020). These files not only assist in identifying malicious, unauthorized activity (OWASP, n.d.; Scarfone & Mell, 2007), but can also be used to reconstruct the step-by-step approach involved in the unauthorized activity (Kent et al., 2006). However, due to the extremely large sizes audit files can reach if required to maintain for any length of time, there has been a concerted effort to identify insignificant data that can be discarded. One approach taken was the use of network flows – the sequence of network packets between a source computer and its destination utilized (Claise, Trammell, & Aitken, 2013; Kerner, 2021; Petryschuck, 2019). However, many IP flow systems exclude the payload to say space, and that removes any ability to trace steps, determine the purpose of the attack, or determine what information is being transferred (Alaidaros and Mahmuddin, 2017).

Another approach was to develop more-efficient compression algorithms with higher compression rates to decrease the file sizes. Unfortunately, to analyze the data for malicious behavior requires decompressing the data back to its original, large size (Loh, Baym, and Berger, 2012). Therefore, a system that not only can reduce the size of the stored files but can reduce the size of the data itself, would be a benefit to cybersecurity professionals.

Chapter 3

Methodology

Overview

Since 2016, internet traffic has been doubling every two years such that it is expected to be 178 billion gigabytes in 2022 (O'Dea, 2020). This growth has had a comparable growth in cybercrime, estimated to approach $8 trillion in 2021 (Embroker Team, 2021). Although there are many forms of cybercrime (FBI, n.d.), this research is mainly concerned with the impact of long-term attacks, such as APTs, and the requirement to maintain audit log files for an extended period of time. The NIST Special Publication 800-39, "Managing Information Security Risk," defines the APT as those attacks that pursue their objective repeatedly over a long period of time; adapt its approach to defense efforts; and are determined to execute its objectives (NIST SP 800-39, 2011). These and other malicious attacks can occur over weeks and even months. Detection requires analysis of the network traffic over the life of the attack, the sheer volume of which can make the problem intractable. To be able to gain insight into APTs and other long-term attacks requires access to audit log files generated over the period of the attacks.

In 1997, NIST published a bulletin that summarized a chapter in the NIST SP 800-12, "Introduction to Computer Security: The NIST Handbook." This publication was devoted to the need and importance of creating audit trails (audit logs) for IT systems

(NIST ITL Bulletin, 1997). The bulletin describes an audit trail as consisting of records of system events concerning the operating system, the users, and the functioning of any applications. The purpose of the audit trail is to assist in determining whether systems have been harmed by insiders, hackers, or even technical problems. The audit trail can be used immediately with access controls to identify possible malicious activity, or maintained for future analysis after a system outage (NIST ITL Bulletin, 1997). To this day, cybersecurity professionals agree that it is essential to collect and maintain audit files (audit trails) in order to ensure accountability, determine who is accessing a network, the applications they are using, any activity that could impact the security posture of the enterprise, and allow for reconstruction of events after a problem has occurred (Marker, 2021; Shopp, 2020; NIST ITL Bulletin, 1997). Although there are many types of audit logs, it is the audit logs comprised of Ethernet packet captures required by network intrusion detection systems (NIST SP 800-94, 2007) that are the subject of this research.

In the Introduction, it was calculated that an organization could generate upward of 90 petabytes of audit log files per year. From a simple storage standpoint, standard compression techniques decrease the size of these files by as much as one-fifth of the original data file size, based on testing of a 6.4 GB Compact Muon Solenoid (CMS) data file (CERN, n.d.) which contained 6500 events (Zheng & Bockelman, 2017). The authors tested different compression algorithms, including versions of the ZLIB (Zip Compression Library), LZ (Lempel-Ziv), and LZMA (Lempel–Ziv–Markov) algorithms against the CMS file. The best results were obtained with LZMA-9, achieving a compression ratio of 5.29; the lowest compression ratio of 2.95 was achieved by LZ4

(Zheng & Bockelman, 2017). A method of compressing and storing a large percentage of the data in offline files, while maintaining keys that can be updated and searched in real time would greatly decrease the resources required to store and analyze audit files and be a valuable tool in fighting cybercrime. However, that ability is not within the scope of this research, only the feasibility of applying genomic compression techniques to network traffic, specifically Ethernet packets with data segments.

Development Considerations

Conceptually, the implementation of genomic compression techniques to network traffic is straight-forward. However, development of any solution is dependent upon an ability to test and validate the solution, so a suitable dataset is paramount. To identify a suitable dataset, it was necessary to identify the scope of network traffic that could be compressed by the new technique, and then the identification of an appropriate dataset.

Network traffic follows a layered service model. The original model development led to the ISO 20000 Open Systems Interconnection (OSI) model, which presents network traffic as seven different layers: physical, data link, network, transport, session, presentation, and application layers (Kurose & Ross, 2007). Packet capture programs, such as Wireshark utilize a 4-layer Transmission Control Protocol/Internet Protocol (TCP/IP) model: 1) network access layer, comprised of the data link OSI layer – the physical layer is not included; 2) internet layer, which is comparable to the OSI network layer; 3) transport layer, identical to the OSI transport layer; and 4) application layer, comprised on the application, presentation, and session OSI layers (Ghosh, n.d.). Not having the ability to test directly on a large network with hundreds or thousands of users, the research was performed utilizing packet capture files. Going forward, all references to layers will be to the appropriate TCP/IP model and not the OSI model, unless explicitly noted.

Although there are methods to replay captured packets (Colasoft, n.d.; Nanni, 2020; Tcpreplay, n.d.), there are potentially significant development issues in their implementation. Besides obtaining, installing, and configuring the replay tool, a method of capturing the replayed packets for testing the compression technique would be required, to the exclusion of all local traffic. These are not particularly onerous tasks, but the only purpose for the replay approach would be to show the ability to process in real time. Since the main intent of this research was to develop a packet compression process utilizing genomic compression techniques, if this proved unsuccessful, the processing in real time would be moot. Therefore, this additional complexity was determined not to be valuable. Instead, simply capturing packet times and comparing to runtimes provided an indication of the ability of the approach to compress packets in real time. Therefore, it was decided to utilize the Npcap library (Npcap, n.d.) to open and read through the data sets, thus providing better control of the process.

Another concern involved the issue of the IPv4 versus IPv6 protocols. According to the Internet Society, in 2018 over 25% of internet-connected network advertised connectivity utilized the IPv6 protocol (ISOC, 2018). However, the research solely relied on IPv4 traffic for one simple reason, the major datasets available for testing are only IPv4 traffic. No matter whether the protocol is IPv4 or IPv6, that only impacts the Internet protocol itself – there is no impact to the transport and application layer protocols. Although synthetic IPv6 data could be generated, or the IPv4 dataset modified for IPv6 format, the difference would have no impact on the success of the genomic compression technique.

As was previously mentioned, packet capture programs work on the TCP/IP, 4-layer model. To maximize compression, parsers were developed for the most-common protocols that include substantive data. The initial development created the main program, the Genomic Network Compression System (GNCS), which opened a PCAP file, read the packets, and would pass to appropriate modules to perform further processing. To determine the scope of the parser development, the next step was to read through the dataset, capture the total of each internet layer and transport layer protocol encountered. Although application layer traffic could be processed individually, since their formats are really variations of the transport layer packets; the differences were not considered significant enough to warrant additional parser development. Thus, the main program consisted of reading packets from the dataset, switching on internet layer protocols, then switching on transport layer protocols to determine the appropriate parser to use. It is in the transport layer parser where the genomic compression technique was implemented.

Parser development involved the application of genomic compressive techniques to minimize the volume of data required to uniquely define the packet. Genomic compression techniques are based on the knowledge that only a small percentage of gene sequences are different between members of the same species (Loh, Baym, & Berger, 2012). Therefore, sequences of the data that are equal to or comparable to a known sequence are identified and replaced with a link to that known sequence along with an edit script identifying any differences. In the case of network traffic, this involved identifying what packet data is common and would lend itself to this type of compression.

At a basic level, this research resembles network flow. In the development of the IP Flow export protocol, Claise, Trammell, and Aitken (2013) defined a network flow as the data stream between source and destination computers. At a minimum, a traditional IP flow consists of the 5-tuple source and destination IP addresses and ports, and the protocol. With protocol-specific parsers, this research only required a 4-tuple key, comprised of the source and destination IP addresses and ports. The basic function of a network IP flow device involves capturing packets, aggregating the flows based on the 5-tuple value, and exporting the flows to a collector (Petryschuk, 2019). Flow devices follow rules to determine when a flow has ended and is to be exported: 1) when the finish (FIN) or reset (RST) TCP flags are set; 2) a configurable time after receiving the last packet of an existing flow has elapsed, typically15 seconds; 3) a maximum time after the flow record was created has been exceeded, default of 30 minutes; or 4) when the flow cache is full (Estan, Keys, Moore, & Varghese, 2004). Although the research compression algorithm utilized packet aggregation, during the process of flow creation, the genomic compression technique allowed for common packet data to be replaced with a link, while the common data was stored offline. However, unlike common archiving compression techniques, this research algorithm is considered a lossy compression technique, in that there is certain information in network traffic that is redundant or not required for analysis. This common data was stored once, and duplicate information discarded.

One issue that can impact future analysis of audit log files is the effect of DHCP which assigns IP addresses to their users' workstations on an as-required basis (Weinberg, 2022). Although systems can be assigned a fixed (static) IP address or they may be repeatedly assigned the same IP address, it is more likely that over a several month period, that a user's workstation will be assigned multiple IP addresses. Therefore, the IP address cannot be reliable, since there is no guarantee that it belongs to the same user as the last time analyzed. In normal network traffic, the source and destination network device media access control (MAC) addresses are available in the network access (data link) layer frame data (Harmoush, 2019). MAC addresses are unique to a single device, so it is theoretically possible to trace a packet back to a single device and thus a single user. This sounds easier than it is. MAC addresses are used for hop-to-hop traffic, meaning the destination MAC address is that of the next "hop" (switch/router) in the path and the source MAC was the previous hop. To reach the ultimate destination, at each hop, the switch must determine the next hop, based on the IP address. Unfortunately, as will be discussed later, some datasets do not contain full network access layer information. Wireshark was used to analyze the candidate test datasets and it was identified that some did not contain any network access layer data, some partial, and some contained a complete network access layer header (Wireshark, 2020). With real network traffic, if the source IP is a member of the enterprise, the source MAC address would represent the source system and could be used in correlation analysis. However, whether DHCP was utilized on the network capturing the test data is immaterial, since the research compression technique was based on the Transport Layer protocol, and there was no further analysis performed correlating the data flows over time.

The main function of GNCS was rather simple, utilizing the Npcap library (Npcap, n.d.). After opening the PCAP file, it was a straight-forward process of reading through the file, packet by packet, extracting the internet protocol from the network access layer and calling the appropriate protocol parser, passing the packet as an unsigned character string. After the last packet was processed, the main program finished by combining the working files into the final output file and saved the processing statistics for comparison to other techniques.

The development of each parser followed a common approach: extract parser-specific (protocol-specific) data and apply genomic compression techniques, where appropriate. For example, in parsing the major transport layer protocols, the payload was a perfect candidate for genomic compression techniques. However, with request/reply protocols, the only possibility would be matching requests and replies – there is no payload. The worse contributors to compression ratios were the unsupported protocols (those without a data segment), where the entire frame/packet was stored as-is, without benefit of the genomic compression techniques.

Development Resources

All programs of GNCS were developed in ISO C++ 2014 Standard for possible portability to other platforms. All programs were standard console applications developed using Microsoft Visual Studio 2022 Professional. The Npcap library was the only non-Microsoft library utilized in development. There were two publicly available tools utilized for development/debugging purposes. The first was the 64-bit Windows version of Wireshark (Wireshark, 2020). This aided in debugging by seeing what data was in a packet and ensuring it was being handled correctly by the program. The other application was the hex editor, HxD Hex Editor (Hörz, M., 2020). This application allowed for opening both the PCAP dataset file to actually see the raw data, and also opening any of the intermediate or final output files to debug any issues with the file structures.

Development occurred on two different systems, a desktop and a laptop. The desktop system is a Dell XPS 8700 with an Intel® Core™ i7-4790 CPU running at 3.60 GHz, 24 GB DDR3 RAM (1333 MHz), and a one (1) TB Samsung 850 EVO SSD running Windows 10 Pro. The laptop system is a Dell XPS 9520 with an Intel® Core™ i9-12900HK CPU running at 2.50 GHz, 32 GB RAM, and a one (1) TB SSD running Windows 11.

Testing required an appropriate Ethernet dataset containing IPv4 packets. There are two approaches possible, use of an existing dataset versus creation of a synthetic dataset. The later was deemed impractical due to the nature of the genomic compression algorithm itself. The various headers required by the Ethernet protocol are short and compact and are therefore of insufficient length to efficiently utilize the genomic techniques. It is the data segment of the application layer that can benefit from compression. To prove the validity of the approach requires a large amount and variety of data segments, something that would be a project itself. Therefore, it was determined that use of an existing dataset would be preferable.

Dataset Analysis

A simple internet search showed there are many packet capture datasets available for various purposes. In many cases, these are small; limited in scope (number of layers and protocols), or through anonymization, the payloads have been stripped. NETRESEC AB is a Swedish software developer that specializes in products designed to perform network security monitoring and network forensics. NETRESEC AB also maintains a comprehensive list of publicly available PCAC network capture files, available from their website (https://www.netresec.com/). Of the numerous available datasets, two groups stood out as promising test dataset candidates:

1. Starting in 2005, the National CyberWatch Center began hosting the Mid-Atlantic Collegiate Cyber Defense Competition (MACCDC), in which students from different universities act as Blue Team members to protect a corporate network from attack by a Red Team made up of volunteers from industry and academia. The network captures for the annual competitions between 2010 and 2012 were downloaded for analysis (MACCDC, 2012).
2. The Institute of Electrical and Electronics Engineers (IEEE) Communications Society (ComSoc) has been a sponsor of the Military Communications Conference (MILCOM) since its inception in 1982. At MILCOM 2016, Bowen, et al. presented a paper that described the development of four synthetic datasets that included cyber exploitations for use in cyber security research (Bowen, et al., 2016). In order to better determine the potential testing value of these datasets, the download page also included a detailed description of five datasets based on the Bowen, et al. (2016) research, as well as supplemental material (https://download.netresec.com/pcap/MILCOM-2016/Datasets_A-E_Descriptions.pdf).

The internet search also provided another source of datasets from the School of Engineering and Information Technology, University of New South Wales at the

Australian Defense Force Academy, Canberra, Australia. The UNSW-NB15 network datasets were compiled by Moustafa and Slay (2015) and are hybrid combinations of real-world network traffic that were modified with some synthetic data for testing intrusion detection systems.

The first program-specific tool developed was a small program that opened a PCAP file, walked through it packet by packet, while compiling the total of network, transport, and application layer protocols within the PCAP file, with special attention to available data segments (payloads). This small program was instrumental in determining the viability of a dataset for compression, as well as identifying the scope of the protocol parsers that would be required.

The application of genomic compression techniques is proposed to be applicable to any network traffic. Since the intent of this research is to show the feasibility of this technique to reduce the size of the network traffic audit/log files, it was beyond the scope of this research to include a wide variety of possible datasets. Rather, the intent was a proof of concept, which requires a dataset representative of real-world network traffic. Another consideration in the evaluation of the candidate datasets is the size and composition of those datasets. As has been discussed, the genomic compression technique needs a sufficient volume of data to be practical – the more data there is, the higher the likelihood of finding similarities. Therefore, datasets with less than 1,000,000 packets were eliminated from consideration.

Appendix B provides a detailed analysis of 68 identified datasets, the results of that analysis, and the datasets that were determined to contain sufficient data to test the feasibility of applying genomic compression techniques to network traffic. Out of those

68 candidate datasets, all but two datasets were eliminated from consideration, and testing was therefore limited to the UNSW-NB15 17-2-2015 dataset (Moustafa and Slay, 2015) and the MACCDC 2012 dataset (MACCDC, 2012). Although neither dataset is solely a capture of normal network traffic, the packets presented were complete and included significant payloads for standard protocols. All further analysis was limited to just the selected datasets. Hereinafter, the UNSW-NB15 17-2-2015 dataset will be referred to as the UNSW-NB15 dataset and the MACCDC 2012 dataset as the MACCDC dataset, unless further distinction is required.

Parser Development

The first level of parsers was determined based on the Ethertype values in the network access layer headers. The Ethertype is stored in the Ethernet packet as a 4-byte unsigned integer (Networx, n.d.), and although there is a theoretical maximum of 65,535 possible Ethertype values, most of these are proprietary (IANA, 2022; Nobel, R., 2018). The more-common registered Ethertypes are 0x0800 (IPv4), 0x0806 (ARP), 0x8035 (Reverse ARP, or RARP), 0x8100 (VLAN – IEEE 802.1Q), and 0x86DD (IPv6) (Networx, n.d.). The actual internet layer parsers developed were determined through the analysis of the candidate datasets. Table B7 in Appendix B shows the occurrences of Internet Layer protocols in the various candidate datasets. In particular, the two selected test datasets (MACCDC 2012 and UNSW-NB15 17-2-2015) had IPv4 and ARP in common, making them candidates for Internet Layer parsers. ARP is a messaging protocol, in that one system requests information from another system or systems, which reply with the requested information. However, with these request/reply message protocols, the only possibility would be matching requests and replies – there is no payload. One aspect of genomic compression techniques is replacing common data segments with links. Therefore, an Internet Layer parser was developed for ARP to determine the feasibility of its application to message protocols.

The next level of analysis was to identify transport layer protocols that possessed a data segment of sufficient size to take advantage of the genomic compression technique. Without developing parsers for every possible transport layer protocol, the analysis was geared toward capturing not only the various protocols, but also the size of the packet (minimum, maximum, and average), as well as the size of any contents beyond the transport layer header. Only those protocols that contain a sufficient data segment were candidates for parser development.

Analyzing the Transport Layer results from Appendix C, Table C8A for MACCDC and Appendix D, Table D5A for UNSW-NB15 shows that there are a total of seven (7) Transport Layer protocols utilized between the two selected test datasets: 0x01, Internet Control Message Protocol (ICMP); 0x02, Internet Group Management Protocol (IGMP); 0x06, TCP; 0x11, User Datagram Protocol (UDP); 0x58, Kerberos; 0x59, Open Shortest Path First (OSPF); and 0x84, Stream Control Transmission Protocol (SCTP). However, except for TCP and UDP, the other protocols not only had an insignificant amount of data, but their occurrences within the datasets were a fraction of one percent. With the research goal of proving the efficacy of the genomic compression technique with network traffic, due to these limitations, these minor protocols were not deemed to be candidates for parsers. Therefore, TCP and UDP were the only Transport Layer protocols proposed for parser development.

As has been discussed earlier, due to the very nature of the proposed research, it was not practical to address all possible protocols, since only packets with a data segment were candidates for compression. Therefore, instead of expending a significant effort and significant resources required to create parsers for these protocols, the unsupported packets were saved as-is, to audit the output file to ensure every packet was processed. However, to evaluate the impact of the genomic compression technique itself, data was gathered pertaining to the packets with usable data to determine the original and compressed record sizes, thus ignoring the negative impact of the unsupported protocols. To support this, another tool that was developed analyzed the completed interim data files and the final output file, to ensure all packets in the PCAP dataset were captured. The final output file is a conglomeration of multiple parser files, and as such, it was important to be able to ensure it was structured correctly and represented all packets captured. Throughout development, the output of this analysis tool also provided debugging information through comparison with the actual packets, as viewed in Wireshark.

Internet Layer Parser

This was a simple parser and was implemented as part of the main PCAP packet read loop. A Npcap function reads each packet and presents it as a hexadecimal character string. It is a simple matter of extracting data according to the packet format. Upon reading a packet, the program walks the string, extracting protocol header information as required. The internet parser stops after extracting the network layer, and calls the appropriate parser, passing the remainder of the unsigned character string for further extraction. Upon return from one of the internet layer parsers, the main program continues the read loop through the PCAP file. If the last packet has been reached, the main program saves all run statistics and consolidates all the interim files into a final compressed file.

ARP Parser

ARP is an example of a protocol without significant data and is discussed here to show how elimination of duplicate data reduced the size of the resultant file. In practice, ARP requires the exchange of messages between the source and destination devices and is used to identify the MAC address associated with a given IPv4 address. The communication is a pair of packets consisting of request and reply packets (The TCP/IP Guide, 2005). RARP utilizes virtually the same format as ARP and performs the opposite discovery – obtain the IPv4 address associated with a given MAC address. The main purpose is for diskless systems to obtain their IP address at boot time. However, that protocol has been superseded by either the Bootstrap Protocol (BOOTP) or the Dynamic Host Configuration Protocol (DHCP) (IONOS Digital Guide, 2019). For those devices/applications that still utilize RARP, the parser will check for the possibility. The difference is in the opcode: ARP utilizes 1 and 2 for request and reply, while RARP utilizes 3 and 4 for the request and reply (The TCP/IP Guide, 2005).

Although the goal of the research was to determine the efficacy of the genomic compression techniques on network traffic, request/reply message formats can take advantage of one of the underlying precepts of the genomic compression technique – reduce duplicate data by referencing once. Therefore, a parser was developed for the ARP packets to identify the magnitude of possible data space savings that could be obtained for request/reply message protocols. The ARP parser maintained an index of packets read from the PCAP file in a balanced binary search tree (BBST). When the reply to a specific request is read, the two associated packets are combined into a single record. Since much of the data is duplicated in the request and reply packets, there really was no data to be archived and linked, so a merged packet was created with additional information to identify the request and reply data points. This combined record was then written to the ARP intermittent file and the request packet was removed from the BBST. Although this record format was not ideal for maximizing compression ratios, any decrease in the amount of data to be retained, by default, will improve compression ratios. This also provided an indication of potential data savings for other request/reply message protocols.

IPv4 Parser

Like the ARP parser, the IPv4 parser is called by the internet layer parser. passing the remaining unsigned character string (packet data). This parser continues reading the packet string and extracting IP header data, the critical data fields being the transport layer protocol and the source and destination IP addresses. At this point, the parser continues processing dependent upon the transport protocol involved. The two most-common transport layer protocols with a significant data segment are TCP and UDP.

Transmission Control Protocol (TCP) Parser

TCP is a connection-oriented, reliable data transfer service. It is connection-oriented because the source and destination systems perform a handshaking to establish a full duplex connection that allows simultaneous communication. It is reliable because the communicating systems are assured that TCP will transfer all data without error and in the correct order (Kurose & Ross, 2007).

After the remainder of the IP header is extracted, the TCP header is next. The first two data elements extracted are the source and destination ports, completing the 4-tuple key. Next the sequence and acknowledgement numbers are extracted, the former required for proper reconstruction of the data if the data is fragmented and covers multiple packets. The next important data point is the header size, sometimes referred to as the data offset since this data point is used to determine where the actual data begins. The next important data elements are the flags. There are nine 1-bit flags, but according to RFC 8311, the first flag, NS, is experimental and thus not germane to this research (Black, 2018). Therefore, that bit can be ignored, and the remaining flags extracted as an unsigned 8-bit integer. Of the eight flags, the only flags that directly impacted the compression process were the Acknowledgement (ACK), Reset (RST), Synchronize (SYN), and Finish (FIN) flags. The other three flags, Congestion Window Reduce (CWR), ECN-Echo (ECE), and Urgent (URG) flags pertain to various aspects to transmission and reception, but do not really impact the compression process, so were ignored. At this point, all pertinent data was available.

The first step was to determine if the packet is the beginning of a new flow or a continuation of an existing flow. The 4-tuple key is stored in a BBST. This structure was chosen for two reasons: 1) being self-balancing, the tree can theoretically hold 2,147,483,649 (2^{31}+1) nodes. This was important, because the initial design utilized an unbalanced binary search tree and walking that tree caused stack overflow errors. Although the default stack size is 32 and can be increased, the second reason was the deciding factor; 2) operations on binary search trees are proportional to the height of the tree. Balanced trees are more efficient than unbalanced trees since they minimize the height, and thus fewer comparisons are required to locate an existing node or where a new node should be inserted (Elgabry, 2017). The key used for the BBST was the source IP, destination IP, source port, destination port, with the IP addresses represented as unsigned, 32-bit integers and the ports as unsigned, 16-bit integers. A node of the BBST consists of the key and a pointer to a packet structure that contains the common data elements of a data flow, plus pointers to the node to the left and the node to the right, and the value of the height in the tree of the node that is used to balance the tree. The last key data point in the packet structure is a link to the data.

The next development issue pertained to collecting and subsequent aggregation of data segments. If the reception of packets at the destination were guaranteed to be in sequential order, a simple list could be utilized to keep track of the data segments, However, order is not guaranteed, so a mechanism to efficiently identify the location of the received data segment is required. To accomplish this, data is stored in a doubly linked list based on the sequence number, maintaining the last insert value as the starting point for the next insert. If packets are received in order, the new data segment is appended to the end of the list. In the case of a packet being received out of order, instead of starting at the beginning and walk the list to find the proper location, which would be required for a simple list, the insert function searches the list from the last insert location until the proper location for the new data segment is identified and it is inserted; the direction of searching based upon whether the sequence number of the latest packet is greater than or less than that of the last inserted packet.

There are several scenarios that identify the beginning of a flow and trigger the end of flow, and the details were covered previously. The SYN flag identifies the beginning of a flow. If there was already a flow in progress for the 4-tuple key, by the very nature of the TCP hand shaking, the previous flow was considered orphaned, closed, and processed as complete. A new flow would be started with the SYN packet. If a RST flag was received, the existing flow was again considered orphaned, closed, and processed as complete. If a FIN flag was received and there was an existing flow, it was considered complete and processed. If a FIN flag was received and no flow existed, the single packet was considered orphaned and processed as complete. Although a timeout period was discussed, since this research was utilizing a finite set of internet packets, the timeout processing was not implemented. Rather, after the last packet was read, all open flows were considered "timed out" and processed as complete.

The remaining impact to the TCP parser was duplicate packets. Pentikousis et al. (2010) performed an analysis on network traces from the National Laboratory of Applied Network Research Passive Measurement and Analysis (NLANR/PMA). This analysis determined that in the various traces in the dataset, retransmissions of TCP packets ranged from 0.48% to 3.6%, with an average of 2.4% (Pentikousis et al., 2010). The parser was designed to identify duplicate packets – retransmissions. Upon detection, the data segments are compared and if there was a difference, the original record was replaced with the retransmission record and a duplicate record of the original packet was created for future reconstruction integrity. A true duplicate record was saved as a very simplified record in which a link to the original packet was saved along with the timestamp of the duplicate, and the remaining duplicate packet data was discarded.

User Datagram Protocol (UDP) Parser

UDP is a connectionless protocol – there is no handshaking before the message is sent. UDP is used by computer applications to send messages (datagrams) to other computers on a network but is considered unreliable since there is no guarantee that the message was received (Kurose & Ross, 2007). Therefore, unlike TCP, there are no sequence and acknowledgement numbers to identify the packet order. Rather, UDP packets are processed in the order in which they are received, regardless of the impact to the receiving program of out-of-order data. All this makes UDP a simple protocol. After the IP header, the UDP format consists of five data fields, the source and destination ports, the UDP length (includes both header and data), a checksum, and the data. Each packet stands on its own, so in the case of a streaming application, such as Voice over IP, there is no way to know when the communications end, other than no further packets are being received. Since a UDP packet could be either a complete message or a streaming service, the various datasets were analyzed for UDP port usage to determine the extent of single packet protocols (message formats) versus a streaming service. UDP packets were collected based on the 4-tuple key discussed above and analyzed for commonality. If there was an insignificant percentage of UDP traffic in a dataset utilizing protocols requiring multiple packets, as opposed to single packet messages, the dataset was deemed not compliant with the requirements for the genomic compression algorithm. Except for the Trivial File Transfer Protocol in the UNSW-NB15 dataset, all other reserved or assigned ports are utilized by messaging protocols. Therefore, the only applicability of genomic compression techniques would be those situations where the data segment of the single packet was sufficiently long to warrant an edit script. In those cases, the genomic compression algorithm would be applied, where feasible. If not successful or the data segment is of insufficient length, the UDP traffic would be saved as is.

Compression Algorithm

The genomic compression algorithm developed by Loh, Baym, and Berger (2012) relied on a known dictionary of the gene being analyzed. As their program progressed, it compared the test gene sequence to the known gene. When a difference was identified, the common portion was replaced with a link to that sequence and an edit script showing the differences. The actual analysis can thus be performed on the much shorter differences (Loh et al., 2012). Although genes are a very large structure, they are all comprised of four basic nucleotide bases, adenine (A), cytosine (C), guanine (G), and thymine (T), and a sequence of three nucleotides makes up an amino acid, of which there are 20. It is how these amino acids are strung together determines the makeup of a protein, which can be hundreds and even thousands of amino acids in length (Genomics, 2022). However, genomic analysis is performed at the nucleotide level, requiring only four different values to compare (Loh et al., 2012).

With network traffic, there is no known dictionary to compare, but it can be created as processing progresses, which was the approach taken by this research. The next issue to consider was the comparison process. Compressed bitmaps are commonly used for database indexes (Wang, Lin, Papakonstantinou, & Swanson, 2017), but considering the potential size of aggregated data segments – a user is downloading a file of hundreds of megabytes in size – bitmaps can become difficult to manipulate. Another possibility would be to compare 4-bit words (nibble), of which there are 15 possibilities in a base 16 system: zero (0) through nine (9) and the alpha characters ‘a’ (or ‘A’) through ‘f’ (or ‘F’). However, the smallest native data type is the 8-bit character. Comparing nibbles would require bit manipulation, making coding less efficient and more complex. Therefore, the logical choice was to compare characters. However, the character data type is a signed 8-bit integer representing values from -126 to 127, so unsigned character strings which represent values from 0 to 255 were utilized. This is also the native data type utilized in PCAP capture files, so no transformation of the raw packet data is required. To support these strings, three small functions were developed to compare, copy, and duplicate unsigned character strings, in addition to more-complex functions to perform approximate string matching.

When a flow was considered complete, the first check was to ensure all possible data was received. With UDP, this was more of a formality since there is no way to know whether any data was lost. With TCP, this was accomplished by walking the string and confirming that a packet sequence number plus its data length was equal to the next packet sequence number. If there is any missing data, the gaps are filled with 0x00 for each missing unsigned character, subject to a threshold on the difference between the sequence of the packet and that of the next available. The threshold is necessary since the datasets potentially contain data specifically designed to gain unauthorized access and might not conform to TCP standards. Since the sequence numbers are unsigned 32-bit numbers, the difference between two sequence numbers could potentially be not just thousands, but millions. It was therefore determined that if the difference between sequence numbers exceeded the maximum data segment of a standard TCP packet, the flow would be split and handled as separate flows.

As was discussed above, a gene is a long string comprised of just four different nucleotides. Genomic analysis entails comparing a subject gene to a known gene and identifying the differences between them. With just four possible values to compare, this is not particularly an onerous task. In 2005, researchers compared the genes of humans to chimpanzees and determined that 99% of the genetic code is similar (Gibbons, 2012). So, if you take two members of the same species, the differences would be less than 1%. With the human genome containing approximately 6.4 billion base pairs (3.2 billion on each strand of the double helix) (Veritas, 2017), the length of the common nucleotides is significant with short segments being different. This is the situation of which the Loh, Baym, and Berger (2012) compression technique takes advantage.

Network traffic cannot take direct advantage of the Loh, Baym, and Berger (2012) compression technique for two major reasons: 1) it is intuitively obvious that the contents of data segments vary from packet to packet, so contrary to genomes, the differences are significant from packet to packet; and 2) genomic research is comparing the differences between two strings, while network traffic requires finding similarity between thousands and even millions of different packets.

The compression algorithm begins after a data flow is closed. Since there is no portion of the data flow that has a direct relationship to the aggregated flow data (nothing links the data to itself), and since a new segment must be compared against the previous, known segments, the data segments are stored in a series of simple linked lists. Since the likelihood of data for different protocols being identical, the linked lists are stored in a balanced binary search tree with the protocol and destination or source being the key. This key was chosen to take advantage of different users accessing the same information.

To conserve system memory, a fragment of the data segment is maintained in the linked list along with a link to the entire data segment which is stored in an interim file. For the purposes of this research, the length of the segment will be determined through analysis of the aggregated data segments. Although in a production environment, that value would probably be configurable, it was decided to use a length close to the mode of the observable data lengths – the data length that occurs most often. For efficiency, the actual size will be divisible by the "memory address size" of the computer; in this case, divisible by 64, for a 64-bit system. Should the fragments match, or the projected edit script and difference be less than the actual length of the known fragment, then the entire segment is retrieved, and the comparison continues. Throughout the rest of this paper, the newly created aggregated flow will be referred to as the "test string" and its fragment as the "test fragment" and it will be compared to a previously stored fragment, referred to as the "known fragment" and its total stored string as the "known string." In genome sequences, the differences are minor compared to the size of the common data, so adding edit scripts and links does not make a significant difference to the overall size of the file. With network traffic and the goal to decrease file sizes, the algorithm had to be cognizant of the length of the data difference and the length of the link and edit script in relation to the overall data length – short segments are best left as is.

Another difference between genome analysis and network traffic is that in genomic analysis, researchers are identifying where fragments differ from the whole sequence. With network traffic, on any given day, a network would experience thousands, perhaps millions of different flows. To analyze every complete flow would not be practical and would most likely require more processing power than would be available. Instead, the data segment will be analyzed using an approximate match algorithm to determine the number of differences between two strings. As discussed in Chapter 2, Review of the Literature, there are several possible algorithms. The Hamming distance is a very efficient measure of the number of differences between two strings – it compares character by character, starting at the beginning of each string until it reaches the last character (Hamming, 1950). It requires strings of equal length, which is not an issue when comparing equal-length data segments. However, it is intuitively obvious that if the comparison starts with the first character of each string and continues character-by-character, the Hamming distance cannot account for common substrings that are offset by even a single character. For example, if 255 characters of two strings that are 256 characters long are identical, but one string is offset by a single leading character compared to the other string, the Hamming distance would be 256 and not a candidate for the genomic compression technique.

The Levenshtein edit distance is also a measure of the number of differences between two strings, but the count it produces includes the number of character insertions, deletions, or substitutions required to convert one string into another (Berger, Waterman, & Yu, 2021). Therefore, in the example above, the Levenshtein edit distance would be 1 by deleting the initial character of one string, making it a better algorithm in determining approximate matches.

Vladimir Levenshtein first presented his algorithm in 1965 in Russian, and then in English in 1966 (Levenshtein, 1966). For two strings a and b of length m and n respectively, the Levenshtein edit distance, d, is calculated as follows:

$$d[i][j] = \begin{cases} |i| & if\ |j| = 0 \\ |j| & if\ |i| = 0 \\ d[i-1][j-1] & if\ a[j] = b[i] \\ \min \begin{cases} d[i-1][j] + w_{del}(b[i]) \\ d[i][j-1] + w_{ins}(a[j]) \\ d[i-1][j-1] + w_{sub}(a[j], b[i]) \end{cases} & otherwise. \end{cases}$$

For $1 \le i \le m$ and $1 \le j \le n$; w_{del}, w_{ins}, and w_{sub} are the weight (cost) of performing a deletion, insertion, and substitution, respectively. In some implementations, the weights of deletion and insertions are one (1) and the weight of substitution is 1 if $a[j] \neq b[i]$, else zero (0) (Cuelogic, 2017; Jokinen, Tarhio, & Ukkonon, 1996). A variation of the Levenshtein edit distance is the Damerau-Levenshtein distance which adds the ability of transposing adjacent characters (Devopedia, 2019). Dmitry Mozzherin (2019) presents a software system that can calculate the Levenshtein or the Damerau-Levenshtein distances and allows for comparing blocks of characters, as opposed to single characters. However, the Edpresso Team from Educative, Inc. presented a simpler approach where all the cost variables are zero – they were deemed to be equal (Edpresso, 2022). The major short-coming of the Levenshtein algorithm is the requirement for a working 2-dimensional array, which, for strings m and n, would be [m + 1] by [n + 1] in size. The matrix starts in the upper left corner with a value of zero and is populated cell-by-cell until full – the edit distance is the value in the lower right cell. If only a few short strings were being compared for commonality, this approach would not be an issue. However, with millions of network packets to search for matches, the total number of calculations necessary again becomes untenable.

Since the publication of the Levenshtein algorithm, there have been several variations proposed. However, in 1975, Daniel S. Hirschberg presented a variation that recursively divides the matrix in half until a series of trivial problems are obtained; the results of which are combined to obtain the final solution (Hirschberg, 1975). There have been various implementations of the Hirschberg algorithm, but they utilize recursion which becomes problematic on very large strings (KokiYmgch, 2018). Therefore, those variations have been discarded in favor of a simpler version released to the public domain in 2015 by Lari Rascu, to be utilized for feasibility.

Since applying an approximate matching algorithm to two strings that could exceed several megabytes in size, the program needs to determine if the two strings are even candidates for full comparison. Besides the Levenshtein and Hirschberg approaches, there is also the possibility of comparing the fragments and identifying the longest common substring (LCS). If the LCS is small in comparison to the length of the fragments, then the number of edit scripts and text would exceed the length of the test string itself. However, a long common substring would leave smaller mismatches requiring edit scripts, and most likely be shorter than the full string. Since the first pass is solely for the purpose of identifying candidates for full comparison, the fastest approach that can determine suitability will be utilized.

Analysis of aggregated data flows was performed to identify the efficiency of the three approaches and whether different approaches should be applied to the fragments versus the entire strings, and possibly based upon the length of the string. The basic program was modified to process all three approaches on the fragments, as well as applying the Levenshtein and Hirschberg algorithms to the entire strings. For all three, the time to process was captured – minimum, maximum, and average time for each pass, as well as the total time to process all packets in the test. For the Levenshtein and

Hirschberg algorithms, the edit distance was also captured and the length of the longest common substring for the LCS algorithm.

10,000 aggregated packets were tested for the three approaches, and the results are presented in Table 2. The LCS algorithm is the fastest of the three and will be the most-efficient method to determine the suitability of two strings for full comparison.

Table 1 – Processing Time Comparison

	Levenshtein Edit Distance	Longest Common Substring	Hirschberg Algorithm
Minimum	288 µs	220 µs	516 µs
Maximum	98 ms	31 ms	44 ms
Average	782 µs	574 µs	1,105 µs
Total All Runs	7,821 ms	5,745 ms	11,050 ms

The Hirschberg algorithm takes almost twice as long to process than does the Levenshtein Edit Distance. This is not unexpected in review of the code. Whereas calculating the Levenshtein Edit Distance requires simple loops, the Hirschberg algorithm utilizes function recursion which entails much more overhead. Although the Levenshtein method requires significantly more memory than the Hirshberg algorithm, with today’s systems, that is not really a concern, compared to time. Therefore, the Levenshtein Edit Distance will be utilized when comparing the full-length strings,

To determine the suitability of two fragments for full comparison, the longest common substring must exceed a minimum threshold. As will be discussed in detail later, each edit script requires a minimum of 17 bytes of overhead per node, and when written to the interim data file, an additional 6-byte header consisting of a link to the known data string and the count of the edit nodes in the script. For LCS to be an indicator of suitability, it is assumed that the common substring is within the middle of the fragment, thus requiring three edit scripts plus the header, or 76 bytes of overhead. For two 256-byte fragments, an LCS equal to the overhead (76 bytes) would be a break-even point. However, the added processing time is not warranted for a gain of zero space, and depending upon the type of data involved, substituting text with integer values could increase the size of the compressed data. Therefore, the threshold will arbitrarily be set to almost double that value, 128 bytes, or half the length of the segment. In a production system, this value should probably be adjustable, since the type of network traffic, and whether it is encrypted or not, would greatly influence the final compression ratio. Regardless of actual value, test segments exhibiting a longest common substring less than the threshold, were maintained as is and written to the interim data file. However, there is one last aspect of string length that must be considered. If a test string is being compared to a known string that is significantly shorter, pursuing that comparison might not be practical. For example, if a test string that is 90,000 bytes long is compared to a known string that is only 2,000 bytes, the edit string would end with 88,000 insertion characters. Effectively, going through this process might only save several hundred bytes and is probably not worth the processing time involved. However, for purposes of this research, where the feasibility of utilizing the genomic compression technique is being determined, no threshold will be utilized – any gain in compression ratio is advantageous.

The code from the Edpresso team was modified to handle unsigned character strings versus ASCII character strings. The code was further modified to provide an output string that symbolically represented the changes required in addition to the classical Levenshtein Edit Distance – the total number of edits required. The output string begins with the first character of the test string and indicates, symbolically, the changes involved to make the test string match the known string. There are four symbols in the output string: the equals sign (‘=’) indicates that the two characters are the same; the exclamation mark (‘!’) indicates that the two characters do not match – character substitution is required; the minus sign (‘-’) indicates that a character deletion is required; and the plus sign (‘+’) indicates that a character needs to be inserted. Since the known string has already been identified as unique and has been written to the interim data file, the goal of the edit script is to identify how to change the known string to match the test string so that the entire test string does not need to be maintained. For example, if the test string is “abcqdef” and the known string is “abcxdef,” the edit string would be “===!===” since except for the ‘q’ and ‘x’ mismatch, the strings are identical. If test string A is “abcqdef” and known string is “abcdef,” the edit string would be “===+===” since an insertion of the letter ‘q’ is required for the known string to match the test string. Lastly, if the test string A is “abcdef” and the known string is “abcqdef,” the edit string would be “===-===” since the deletion of the letter ‘q’ is required for the known string to match the test string.

The candidate datasets are comprised of multiple PCAP files that are individual, contiguous pieces of a single, large PCAP file. The MACCDC 2012 dataset consists of 17 individual files for a total of 16.8 GB of raw data; the UNSW-NB15 17-2-2015 dataset consists of 27 individual files for a total of 49.7 GB of raw data. Therefore, the link to the location of a data segment in the interim file required an unsigned 64-bit integer, since a 32-bit unsigned integer can only reference a maximum of 4,294,967,295 bytes. To identify where the difference occurs in the test data segment, a 32-bit unsigned integer was chosen over the 16-bit version, since the latter could only handle a maximum data flow of 65,535 bytes. From personal experience, downloading files exceeding 65,535 bytes is very common, so the 32-bit variable will ensure data integrity. The process of creating the scripts will be iterative, creating an interim string of edit scripts. The first data element is the 16-byte index to the original data on the interim datafile, followed by edit segments, each of which utilizes the following structure:

```
struct {
        char                // [edit type]
        unsigned int        // [start location]
        unsigned int        // [data length]
        unsigned char *     // [mismatch data]
}
```

The edit type takes two values: ‘T’ denotes there is a data mismatch, and the test data is contained in the script. In this case, the mismatched data field will be included and will be the test data not found in the known string as an unsigned character string; ‘K’ denotes the known string and since there is a link to that string and the location and length are provided, there is no need for the fourth structure element, so it will be ignored for known strings.

Output File Structure

There are several interim files utilized by the various parsers to hold processed data. These served several purposes: 1) processed data could be removed from active memory and any BBST and lists, thus improving processing speed, and memory usage; 2) having segregated interim files facilitated keeping track of the different types of completed flows; and 3) the interim files provide a simplified process for combining the final output file. Although a common record format could be developed to handle all types of interim structures, the use of record formats specific to the type of parser was deemed the most space efficient. The final file format starts with a header section that provides the location of the specific record type, the number of those records, and their length. None of the records contain the data segments, just a link (index) to the segment or the edit script. Where the record length was not fixed, such as when unsupported protocols were encountered, there is no standard record size. Rather, the individual record size is prepended to the unsupported record as it is written. The end of the file contains a summary of the PCAP files processed, including the file and packet header information, and the file name. The final file format header is as follows:

1. Location of file header data.
2. TCP completed flow records location.
3. TCP completed flow record count.
4. TCP completed flow record format length.
5. TCP orphan record location.
6. TCP orphan record count.
7. TCP orphan record format length.
8. TCP duplicate packet location.
9. TCP duplicate packet count.
10. TCP duplicate packet format length.
11. UDP record location.
12. UDP record count.
13. UDP orphan record location.
14. UDP orphan record count.
15. ARP completed record location.
16. ARP completed record count.
17. ARP completed record format length.
18. ARP orphan record location.
19. ARP orphan record count.
20. ARP orphan record format length.
21. ARP duplicate packet location.
22. ARP duplicate packet count.
23. ARP duplicate packet format length.
24. Ignored TCP protocol record location.
25. Ignored TCP protocol record count.
26. Unsupported Network Layer protocol record location.
27. Unsupported Network Layer protocol record count.
28. Unsupported Internet Layer protocol record location.
29. Unsupported Internet Layer protocol record count.
30. Unsupported Transport Layer protocol record location.
31. Unsupported Transport Layer protocol record count.
32. Data segment location.

33. Data segment count.

The actual interim file contents are then written to the final file, in the order shown in the header, culminating with the PCAP file information, and storing its start location at the beginning of the file. For efficiency, the TCP completed records are not written to an interim file. Rather, they are written directly to the output file, since they are the first true records written.

The compression method to apply to the final file can vary, dependent upon whether speed or final size is to be optimized. Zheng and Bockelman (2017) tested 10 different compression algorithms against a 6.4 GB file. If speed is more important than compressed file size, the LZ4 algorithm exhibited the fastest overall compression times. If the smallest compressed file size is the objective, then LZMA-9 provided the highest compression ratio, reducing the 6.4 GB raw data file to 1.21 GB. If the goal is a combination, then one of the ZLIB algorithms would suffice. The candidate dataset files are downloaded as archives consisting of compressed individual files. Therefore, it was decided to forego the final compression of the output file, since the test dataset would not be a single compressed file, and it would not be representative to compare compressed files. Instead, the size of the interim data file comprised of data strings and edit scripts will be compared to the size of the total of all raw data, to determine the potential compression ratio.

Data Analysis

On the surface, analyzing the results of a compression technique is simple – what compression ratio was obtained and how long did it take to compress or decompress the dataset compared to other compression techniques. However, that is only the bottom-line analysis. Since different compression algorithms are optimized for size or speed, and that is therefore based on the access requirements for the data, it was determined that the best analysis would be comparison of raw data sizes to the final data sizes after application of the genomic compression technique. A secondary reason is that the results of any one file compression algorithm is dependent upon the type of data in that file, so comparing the compression ratios and processing time to compress and decompress could be skewed just between the raw dataset and the final genomic output file.

In addition, since this research was a proof of concept of applying genomic compression techniques to network traffic, there were protocols at the Network, Internet, and Transport layers that were ignored, and not processed. Therefore, it was determined to compare just the raw data of supported protocols to the compressed records. This would be a true representation of the potential for improving the size of a compressed file.

Summary

Since 2016, internet traffic has doubled approximately every other year (O'Dea, 2020), as has computer crime (Embroker Team, 2021). In 1997, NIST published an IT Laboratory Bulletin outlining the needs for computer audit trails/audit logs, and that practice is now considered standard, including maintaining log files for one or more years (Marker, 2021; Shopp, 2020). As was calculated in Chapter 1, an organization could generate 90 petabytes or more of network traffic annually. Although there are multiple archiving/compression applications available, they create static, fixed compressed files that need to be decompressed for the data to be accessible.

Genomic research requires access to extremely large datasets. If these datasets need to be uncompressed to allow analysis, the resources required could be extreme and possibly cause major delays in any analysis. Loh, Baym, and Berger (2012) developed an approach to compression which identified the common parts of a gene and replaced them with a link and an edit script of the differences. In this way, they could analyze a large gene database with significantly fewer resources and in a significantly shorter period. This research applied those same basic principles to network log files – common data (packet/frame payloads) was stored offline and replaced with a link to the edit script (structure) for the data in the record associated with the 4-tuple source and destination IP addresses and ports.

This research is limited to determining the feasibility of applying genomic compression techniques to network traffic, with the uncompressed final output being compared to the original, uncompressed dataset. As is, a compression algorithm that would allow packet capture and compression in real time at higher compression ratios could be a valuable tool in fighting cybercrime. However, extending the approach to allow for compressing and storing a large percentage of the data in offline files, while maintaining keys that can be updated and searched in real time would not be that difficult to implement.

Chapter 4

Results

Introduction

In chapter 3, the method of extending genomic compression techniques to computer network traffic was provided. However, developing the actual steps required to apply this technique to network traffic was more evolutionary in nature. Researching the various internet protocols and the formats is straight-forward, but understanding all the vagaries of their implementation consumed a major portion of the development time. Without the availability of a large internet traffic capture file representative of the real world, a search was performed to identify candidate datasets.

As has been discussed, the genomic compression technique requires the packets to contain a data segment/payload. This immediately eliminated some datasets that were published as being anonymized for privacy with the data segments being removed. To eliminate other candidates, a program was developed to read through a dataset and identify the various protocols and the length of additional data. Many datasets were comprised of a single capture file, that had no or an insignificant amount of post-header data and did not require further analysis. However, there were several datasets that met this first level of evaluation as was discussed in Chapter 3, Methodology and in more detail in Appendix B, which provides the full detailed analysis of the candidate datasets.

As was shown, out of the 68 candidate datasets, two were chosen for test: the MACCDC 2012 and UNSW-NB15 17-2-2015 dataets.

Parser Development

As was discussed in Chapter 3, the UNSW-NB15 dataset was comprised of only two Internet Layer protocols, ARP and IPv4, with the bulk of the packets being with IPv4. Although the MACCDC dataset was comprised of five different Internet Layer protocols, ICMP, Kerberos, and IGMP comprised only approximately 1% of the total, while the remainder were ARP and IPv4 protocols. Since the program was developed for use with other possible PCAP datasets, a contingency was added in case there were unsupported Internet Layer protocols. In this case, the entire packet would be saved to an interim file, prepended by the timestamp of the packet and the actual captured length of the packet.

ARP Parser

The Address Resolution Protocol (ARP) is rather simple, comprised of a request packet and a reply packet. Each packet is identical in structure with the main differences being the opcode and the requested/missing data. The basic ARP format is shown in Figure 1 – the horizontal axis is in bits. The depicted format is representative of Ethernet or IEEE 802 networks with which the hardware addresses are six (6) bytes long and the protocol address lengths are four (4) bytes long, representative of IPv4 packets. The other field

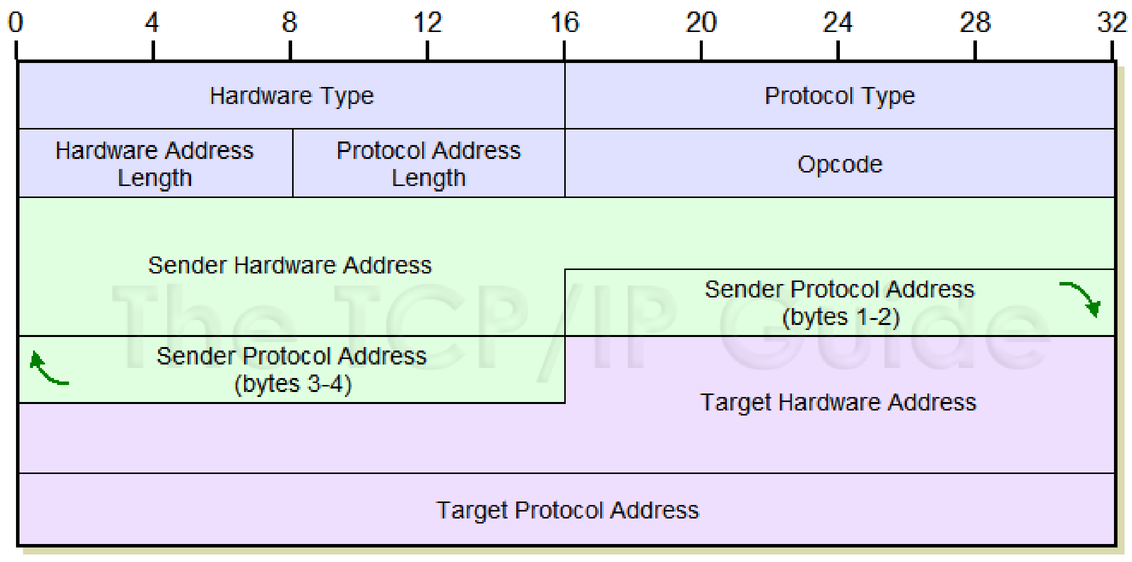


Figure 2- ARP Packet Format (The TCP/IP Guide, 2005)

sizes are as depicted in Figure 1, for a total packet size of 28 bytes per packet, or 56 bytes per request/reply pair for the basic ARP format. However, the PCAP capture format includes a 16-byte header record for each packet, making the total for the request/reply pair as 88 bytes. Since there is no data segment in the ARP format, the ability to apply genomic compression techniques is limited to ignoring common (duplicate) data elements.

The actual program was not as straight-forward as matching request/reply pairs. During processing, ARP packets are stored in a balanced binary search tree using a key of the combination of the source and destination IP addresses. As an ARP packet is received, the tree is searched for existence of the pair. From the results of that search, there are several possible scenarios:

1) Complete Pair. If the search succeeds, the opcode of the current packet is checked to confirm it is a reply. If the request packet in the search results has an opcode

of 1 or 3 and the current packet opcode is 2 or 4 respectively, then this is a matched request/reply pair and is written to the complete ARP interim file.

2) Duplicate. If the search of the tree was successful, but the current packet is identical to search result packet, this is considered a duplicate. In this situation, a record will be written to the duplicate ARP interim file.

3) Orphan. There are several scenarios that result in declaring a packet an orphan. A reply could be received before a request, possibly caused by the reply being a retransmission of a previous reply which had completed a request/reply pair; there could have been an unknown network issue occurred where the request was not captured; the current packet was toward the beginning of the capture that didn't start after the request would have been captured; or a reply is not received. This can occur due to there being no systems available to reply at the time, necessitating sending another request, or it is the end of the run and the reply has not yet been received for any remaining request packets in the search tree. In these scenarios, the packet is considered an orphan and a record will be written to the orphan ARP interim file.

Complete ARP Pair

An ARP pair is considered complete when the search is successful and the current packet opcode is appropriate for the request packet, thus completing a request/reply pair; the results are written to the interim file. The structure of the final combined ARP request/reply pair is as follows:

```
typedef struct {
        struct timeval        rqstTS;        // time stamp for request packet
        struct timeval        rplyTS;        // time stamp for reply packet
        unsigned int32        rqstAddr;      // request IP address
        unsigned int32        rplyAddr;      // reply IP address
        unsigned int16        hType;         // hardware type
        unsigned int16      pType;          // protocol type
        unsigned int16      rqstOper;       // Operation requester is performing
        unsigned int16      rplyOper;       // Operation reply is performing
        u_char              rqstMAC[6];     // request MAC address (6 bytes)
        u_char              hLen;           // length of hardware address
        u_char              pLen;           // length of internet layer protocol
        u_char              rplyMAC[6];     // reply MAC address
}
```

The two timestamp values are obtained from the respective header records, and they can be used in any reconstruction efforts. This structure is 46 bytes in size compared to the 88 bytes required for the request and reply packets plus their PCAP headers. Since ARP packets are a minor subset of the UNSW-NB15 dataset, it is not possible to compare compressed sizes, but it is safe to assume that at a minimum, the raw percentage savings would be representative of the compressed savings.

Orphan ARP Packets

An orphan is a single request or reply packet that cannot be matched with its pair. Except in the situation where the orphan is not identified until the end of the run, orphans are written to the interim file upon detection in a structure different than the actual packet layout to account for byte alignment.

Duplicate ARP Packets

As was discussed, if a request packet is captured that is identical to an existing request packet, it is considered a duplicate. However, unlike orphan packets, being a duplicate, only the timestamps are stored – that information can be used to link back to the original packet for reconstruction of the remaining information.

IPv4 Parsers

The basic functioning of the IPv4 parser was discussed in Chapter 3, Methodology. Its main purpose was to read enough of the packet data to identify the transport layer protocols and send the packet data to the respective parser. The detailed analysis presented in Appendix B demonstrated that both the UNSW-NB15 and MACCDC datasets contain significant transport layer data that could benefit from the genomic compression techniques. The Transmission Control Protocol (TCP, protocol number 0x06, decimal 6) and the User Datagram Protocol (UDP, protocol number 0x11, decimal 17) comprise at least 98% of packets observed in the transport layer. If the transport layer protocol was not supported, the original packet was written to the unsupported transport layer protocol interim file, prepended by the packet timestamp and length of the packet.

Similar results were obtained from the utility program on analysis of the UDP protocol. Appendix E, Tables E3A, E3B, and E3C for the MACCDC dataset and E4A, E4B, and E4C for the UNSW-NB15 dataset and tables C11A and C11B provide the details for the top 10 ports identified in the two datasets. As with TCP, the distribution of ports was different for the two datasets, but like the results with TCP, the format of the UDP packets was not compromised, so all packets were included in the analysis.

Transmission Control Protocol (TCP) Parser

This was the most-utilized parser, since 98.3% of the UNSW-NB15 dataset and 99.9% of the MACCDC dataset consisted of TCP packets. After the IPv4 parser determined the packet to be TCP, it called the TCP parser for further processing, which continued to read the packet data where the IPv4 parser left off.

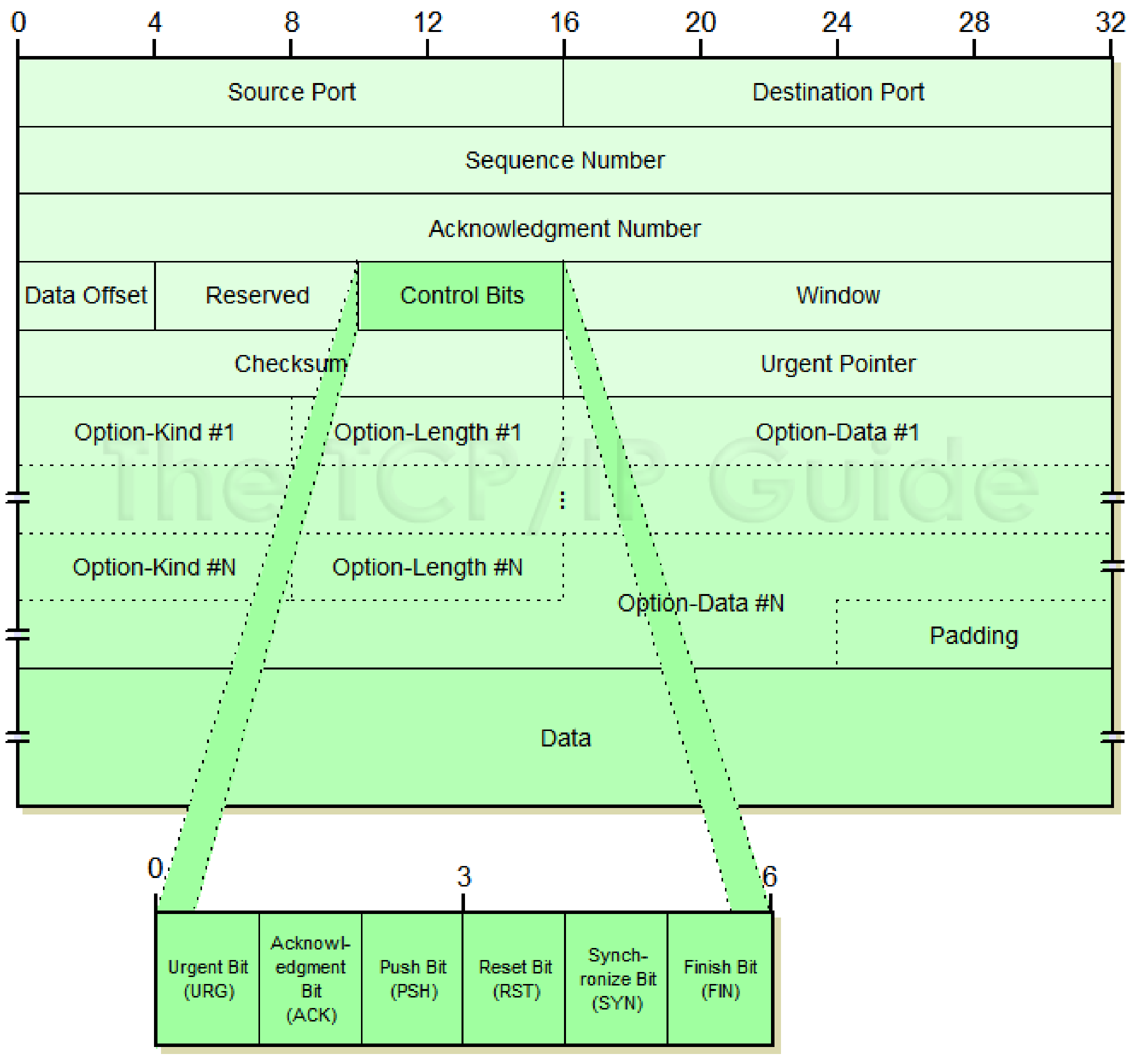


Figure 3 - TCP Segment Format (The TCP/IP Guide, 2005)

The TCP/IP format is shown in Figure 3 and is very flexible with several optional fields. However, since the purpose of the research was to determine the feasibility of applying genomic compression techniques to network traffic, any optional fields were either considered part of the TCP header and thus ignored, or if not included in the TCP header length, they were considered a portion of the data segment.

After the network and Internet layer headers are processed, the TCP Parser takes over. The first fields extracted are the source and destination ports with the low port designated as the TCP protocol as discussed above, followed by the sequence and acknowledgement numbers, and the Controls Flags. The Window, Checksum, and Urgent Pointer fields are bypassed to reach the data segment.

In Chapter 3, TCP was introduced as a connection-oriented, reliable data transfer service. As such, there is bidirectional communication between the source and destination computers to ensure the transferred data is complete. Briefly, this is accomplished through inclusion of a sequence number on each packet that is incremented according to the size of the previous data segment, and values of the control flags.

The TCP parser is centered around two distinct structures. The first is a balanced binary search tree (BBST), utilizing a 4-tuple key of the source and destination IP addresses and ports. This tree stores the unique, unidirectional data flows occurring between two systems. After all pertinent data is extracted from a packet, the TCP parser attempts to insert the new 4-tuple key into the BBST. There are several possible outcomes. If the 4-tuple key is not found, the program checks the status of the control flags. If the FIN flag is set, that indicates the communication is finished. Being that the 4-tuple is not in the BBST indicates that this is probably a retransmission of a previous flow that has been completed. Therefore, this packet is considered an orphan and processing returns to the PCAP reader. The other possible flag is the RES or reset flag, which indicates that the communications is being reset and the flow is thus stopped. As with the FIN flag, since the 4-tuple was not on file, this packet is declared an orphan and processing is returned to the PCAP reader. In both these cases, the packet is written to an interim file using a simple format: the timestamp, the length of the packet, and the full packet itself. The last scenario is the 4-tuple is not found and neither the FIN nor RES flags are set. This indicates it is the beginning of a new flow, so the 4-tuple is added to the tree and processing is returned to the PCAP reader.

If the 4-tuple key is found, there are two possible outcomes. If the SYN or synchronize flag is set, that would normally indicate the beginning of a new flow. Therefore, the current packet is either a packet retransmission or a FIN packet was not received and the existing flow is actually complete and the new packet indicates a new flow should be started. In this case, the existing flow must first be completed before inserting the new flow into the BBST. Processing completed flows will be covered later. If the SYN flag is not set, the packet is simply marked as a duplicate, subject to further analysis.

After the 4-tuple is inserted into the BBST, the second key structure comes into play. This is a doubly linked list that maintains the data segments in order according to the sequence numbers. Each list is unique to the 4-tuple flow, so the primary key is just the sequence number of the packet. As packets with common keys are identified during the BBST insertion process (the 4-tuple was found), the next step is to attempt to insert into the sequence list. If the sequence number is not found, a new node is added to the list which contains packet-specific fields, including the full data segment. If the sequence number is already on file, this is considered a retransmission. The next check is to determine if the data segment of the retransmitted packet is different. If not, the packet is declared a duplicate. However, if there is a difference, the new packet replaces the existing packet which is now considered a duplicate. Packets so identified as duplicates are written to the duplicate interim file, using a similar format as the duplicate ARP file: the packet timestamp and the frame numbers of the original and duplicate packets. The

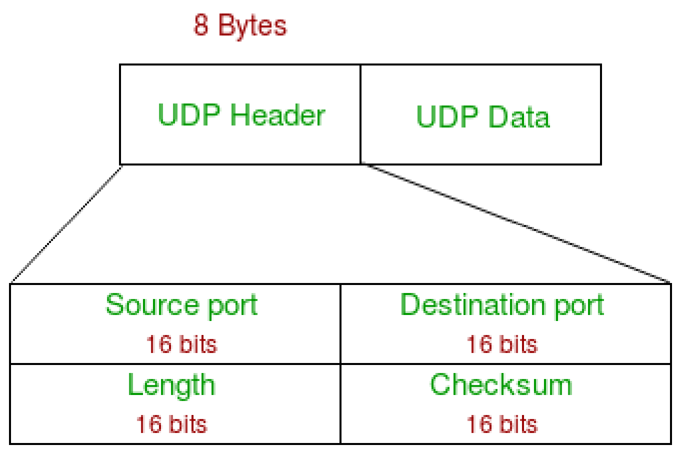


Figure 4 - User Datagram Protocol (UDP) Header Format

last processing on the sequence structure is to determine if the FIN flag is set. If so, this indicates the last packet in the flow, so the flow is completed, and written to file.

User Datagram Protocol (UDP) Parser

As was discussed in Chapter 3, UDP is a connectionless communication protocol in that the packet is sent without concern as to whether it will be received. There many uses of UDP for communications, such as Domain Name Service (DNS port 53), Dynamic Host Configuration Protocol (DHCP port 67), NetBIOS (ports 137 - 139), and Simple Network Management Protocol (SNMP port 161). All these protocols are single-packet or request/reply message protocols and will not require aggregation of data flows. If the single data segment meets the threshold for application of the genomic compression algorithm, it will receive further processing. If not, the packets were saved as is, prepended by the timestamp and the packet length.

UDP is a transport layer protocol that runs over the internet protocol (UDP/IP). Figure 4 shows the UDP header format, which resides in the packet immediately following the IP header. The UDP header is eight (8) bytes in length, consisting of the source and destination ports, the length of the UDP structure including both the header and the data, followed by a checksum. After the checksum is the variable length data segment (limited by the Ethernet packet size restrictions).

To understand the scope of usage by UDP, the utility program analyzed the MACCDC and UNSW-NB15 datasets to count occurrences and data usage of UDP ports. Analysis of the MACCDC dataset showed that DNS (port 53) and NetBIOS Name Service (port 137) comprised 76% of the 549,510 UDP packets and 57% of the total data usage (Appendix E, Tables E3A and E3B). Analysis of the UNSW NB-15 dataset showed that DNS (port 53) and Open Network Computing Remote Procedure Call (ONC RPC, port 111) comprised 74% of the 1,430,377 UDP packets and 64% of the total UDP data usage (Appendix E, Tables E4A and E4B). These three protocols as well as most of the remaining top 10 UDP protocols are message or request/reply format, in that the data is linked to either the single packet or the pair and is not really a candidate for aggregation and application of the genomic compression technique.

Although File Transfer protocols usually entail multiple packets and thus lend themselves to the advantages of the genomic compression techniques, only the Trivial File Transfer Protocol (TFTP) was identified in the UNSW NB-14 dataset, and only comprised 0.1% of the UDP packets; TFTP was not in the top 10 of the MACCDC dataset. With UDP only making up 1.6% of all packets in the UNSW-NB15 dataset and 0.08% of MACCDC dataset and streaming and file transfer usage only accounting for less than 0.03% of either the dataset, it was determined that the added complexity of attempting to create data flows for UDP traffic was not warranted, and all UDP traffic was handled as single-packet communications.

Genomic Compression Algorithm

As discussed in Chapter 3, the implementation of the genomic compression algorithm to network traffic required a multi-step process. The sheer volume of network traffic and the unknown nature of network packets required an approach that maximized efficiency. With genomic analysis, a gene segment is tested against a single known gene. With network traffic, instead of the one-to-one analysis of genomics, there is a many-to-many analysis with network packets.

Table 2 – Transport Layer Protocol Usage

MACCDC Dataset			UNSW NB-15 Dataset		
Protocol	Packets	Data (bytes)	Protocol	Packets	Data (bytes)
0x06–TCP	68,357,197	13,593,753,312	0x06–TCP	86,029,251	45,078,624,185
0x11–UDP	560,934	41,212,397	0x11–UDP	1,430,389	111,862,194
0x01–ICMP	526,103	13,117,305	0x59–OSPF	13,766	1,276,200
0x58–EIGRP	303,549	12,141,960	0x84–SCTP	1,856	1,497,008
0x02–IGMP	1,946	26,808	0x01–ICMP	1,594	824,684
Totals:	69,749,729	13,660,251,782	Totals:	87,480,078	45,194,340,111

Table 2 summarizes the usage of transport layer protocols in the two datasets. Although TCP is the dominant Transport Layer protocol in both datasets and UDP is the second most observed protocol, from there, the two datasets differ. With the UNSW NB-15 dataset, the other three transport layer protocols are insignificant and ignored for processing. However, the MACCDC dataset was not so obvious. UDP was still second with 560,934 packets, but Internet Control Message Protocol (ICMP) was a close third with 526,103 packets, followed by the Enhanced Interior Gateway Routing Protocol (EIGRP) with 303,549 packets. Comparing data bytes and the difference is wider, with UDP having 41,212,397 bytes, followed by ICMP with 13,117,305 bytes and EIGRP with 12,141,960 bytes. ICMP is a request/reply message protocol with the default data payload being 32 bytes with a maximum of 576 allowed (ICMP, 2018). The MACCDC dataset analysis showed that the minimum data payload was 8, and the maximum was 544, with an average of 24 bytes (Appendix B, Table B8). However, being a request/reply message format, the protocol could not take advantage of the genomic compression technique, so was bypassed and saved as is. EIGRP is a network protocol for routers to communicate using a series of message packets request and reply messages used to provide authentication services (Sheldon, 2021). Therefore, despite the frequency of ICMP and EIGRP in the MACCDC dataset, the lack of UDP Application Layer protocols that support significant data, the only protocol that lends itself to data aggregation and compression using the genomic techniques is TCP.

The first step in the process was to eliminate as many packets as possible. This was primarily accomplished through the identification of duplicate data, ARP protocol packets, and less-frequent protocols that were not fully processed; none of these packets are candidates for further processing. Since dataflows, which include file downloads and uploads, can easily be thousands and even millions of bytes in size, another contributor to efficiency was the determination to initially compare just a segment of a packet flow. Therefore, the first 256 bytes of a packet flow is maintained for comparison, rather than the entire flow. Although there is no proof, from years of personal experience, it is the opinion of the author that if the first 256 bytes of two aggregated data flows have no similarity, it is unlikely that the remaining data in the flow will have any similarity. The genomic compression algorithm thus begins with aggregating the packet flow after a triggering event is identified and then extracting the first 256 bytes from the aggregation, or the total flow if less than 256 bytes.

In Chapter 3, Methodology, there was significant discussion concerning determination of the suitability of a data flow for application of genomic compression techniques. It was shown that applying a longest common substring algorithm to 256-byte segments of a test flow and a known flow would provide a good indication as to the likelihood that the two flows exhibited enough similarity to warrant performing the full comparison. Further, it was determined that the Levenshtein edit distance was more efficient for processing time when compared to the Hirschberg algorithm, so it became the algorithm of choice.

The actual algorithm utilized is a modification of the Levenshtein edit distance to include the creation of an edit string that identifies the changes necessary to convert the known data flow into the test data flow. This is created by walking the matrix created by the Levenshtein algorithm from the bottom right cell back to the top left cell, using the reverse of the algorithm to determine whether characters in the flow are equal, not equal, a character insertion, or a character deletion is required, represented by the '=', '!', '+', and '-' characters in the edit string, respectively. Since the edit string is created by walking the matrix backwards, the edit string is reversed for use in creating the overall edit script.

The process starts with the beginning of each flow and the edit string, with separate position counters for each of the three. Unlike genomic compression techniques, the purpose of this research is to develop a more-efficient network traffic compression algorithm and therefore, the research is not interested in analyzing the subtle nuances of the different characters in each string. Rather, it is solely to identify how many characters are required from each string to be able to reconstruct the original. The edit string is analyzed symbol-by-symbol and the edit string incremented after each analysis. If a '=' or '!' symbol is encountered, the two flows are equally incremented. If a '+' sign is encountered, that indicates a character must be inserted into the known flow, so that position counter of the test flow is incremented, but not that of the known flow. Conversely, the '-' symbol indicates a character must be removed from the known flow, so the known flow position counter is incremented, but not that of the test flow. In this manner, the edit string and flows are walked until the end of the shorter flow is reached, and the remaining characters warrant either insertion or deletion, depending upon which of the flows is the shorter.

The other key aspect of the edit script creation is determination of the starting and stopping points of the individual scripts. Those are determined by contiguous '=' symbols in the edit script. When a string of equal characters is encountered that is of a minimum length (discussed later), the progression is halted, and two edit script nodes are saved. The first node would represent the test flow data to save, and the second node would indicate the starting position and length of the known data flow to save. For example, if there are a combination of symbols encountered through the first 65 bytes of the test data flow, followed by 245 bytes of common data ('=' symbols in edit string), the two nodes could be:

```
First node:     edit type: 'T'
                Starting location: 0
                Data length: 65
                Data: "2a2031204558495354530d0a61303033204f…"

Second node:    edit Type: 'K'
                Start location: 54       (accounts for insertions and deletions)
                Data length: 245
```

As indicated in Chapter 3, Methodology, each node has an overhead of 17 bytes, plus an additional 6-byte header when the edit script is written to file. In the example above, while the raw test data flow would be 310 bytes (65 bytes plus 245 bytes), these two nodes reduce that to 124 bytes (2 nodes at 17 bytes each, plus 65 bytes of test flow data, plus the 6-byte record header). During actual processing, the nodes will actually be handled in a linked list and collapsed into the sizes identified upon completion of script creation and writing the output to the interim data file.

Final Program Results

ARP Results

Both test datasets contained ARP packets, but to different degrees. The MACCDC dataset contained 101,294 ARP packets, while the UNSW-NB15 dataset contained 12,081 packets. The distribution of completed request/reply pairs versus duplicate packets and orphans also was different between the two datasets. The MACCDC dataset had almost as many duplicate packets as completed packets, while the duplicate packets in the UNSW-BN15 dataset were approximately one-eighth of the completed packets.

Table 3 - ARP Results – Full Run

	MACCDC 2012 Dataset				
	Packets	Packet Sizes w/Headers	Records	Record Sizes	Size Savings
Completed	52,518	2,310,792	26,259	1,207,914	47.73%
Orphans	1,030	45,320	1,030	45,320	0.00%
Duplicates	47,746	2,100,824	47,746	763,936	63.64%
Totals	101,294	4,456,936	75,035	2,017,170	54.74%
	UNSW-NB15 17-2-2015 Dataset				
	Packets	Packet Size	Records	Record Sizes	Size Savings
Completed	10,688	470,272	5,344	245,824	47.73%
Orphans	3	132	3	132	0.00%
Duplicates	1,390	61,160	1,390	22,240	63.64%
Totals	12,081	531,564	6,737	268,196	49.55%

Table 3 provides the results for processing of ARP packets using GNCS for the two datasets. Although there is no real data segment with the ARP packet format, through elimination of duplicate data elements and no longer requiring the PCAP packet header, the savings of the size of the raw data for completed pairs is 38.64% and 47.73% for the MACCDC 2012 and UNSW-NB15 17-2-2015 datasets, respectively. If orphan and duplicate packets are included, the overall size savings is approximately 50% for both datasets. Although it was not part of this research, it is safe to assume that similar decreases can be expected with the other common message (request/reply) formats.

TCP Results

As has been discussed, outside of ARP, the only protocol that contained sufficient data to test the feasibility of the genomic compression techniques was TCP. To analyze the effectiveness, only TCP packets were included in the analysis since inclusion of unsupported protocols would negatively skew the results. This exclusion also included any detected duplicate and orphan packets. The structure used to save duplicates would improve the perceived space savings, but that would be an artificial improvement. Orphan packets and flows are incomplete, so also would not properly contribute to the results of the genomic compression algorithm. Thus, only complete TCP flows were analyzed. This is the most representative comparison of the impact of applying genomic compression techniques.

Both PCAP files and the genomic compression algorithm have overhead. Each PCAP packet is prepended with a 16-byte header. The packet length itself is variable depending on the Ethertype which determines the length of the Layer 2 header – between

14 and 18 bytes – and the length of any associated data. The Internet and Transport layer headers are 20 bytes each for IPv4 packets, for a total of 54 to 58 bytes plus any associated data. The MACCDC dataset utilizes a standard Ethernet Layer 2 header of 18 bytes, so the overhead per packet is 58 bytes plus the PCACP header, for a total of 74 bytes. The UNSW-NB15 dataset utilizes the Linux "cooked" capture header of 14 bytes, making the total header overhead 70 bytes for these headers.

The output file structure for TCP flows in this research consisted of two components, a summary record and a data flow reference. The summary record consists of a 52-byte header plus 17 bytes per packet contributing to the flow. In addition, the dataflow itself has an eight-byte header consisting of two 4-byte unsigned integers, the data flow index and the length of the flow. The PCAP representation of a completed flow consisting of two packets but no data, would require a minimum of 144 bytes. The same two packets would be represented by 86 bytes, for a savings of 58 bytes. If there is data, this would increase to 94 bytes due to the eight-byte data header. However, as the number of packets in a data flow increases, the total of the standard PCAP packets in the flow would increase by up to 74 bytes per packet in the flow. Kim, et. al., (2004) calculated that the average packet flow consists of 28 packets. Therefore, not counting the data itself, a 28-packet flow would require up to 2,072 bytes of overhead, compared to 542 bytes for the genomic compression algorithm, an effective savings of 1,530 bytes for a space savings of 74%. Although this appears to be significant, this has to be compared to the total flow size including data. In the case of the 28-packet flow, if each packet carried the maximum data – 1518 bytes – then the total of the standard PCAP capture would approach 44,600 bytes. If the data is unique and no comparison found, then the genomic compression algorithm would require 43,054 bytes for a space savings of 3.5%. The real savings is dependent upon the impact of the genomic compression algorithm.

Although the program was compiled for speed, there was no effort to improve efficiency through programming techniques that take advantage of multi-threading or multiple cores. With packets being processed sequentially, the time to process a packet increased as the total number of packets increased. The result was such that processing the complete MACCDC 2012 or the UNSW-NB15 17-2-2015 datasets required up to a week of processing time to complete. Therefore, it was decided to process incrementally larger segments of each dataset to determine if there is a threshold beyond which further processing would not improve the results. The starting point was arbitrarily chosen to be 1,000 packets and each subsequent run increased by a factor of ten until 10,000,000 packets were processed. Since the next factor of ten level would be 100,000,000, which is larger than either dataset, the last sample was set to 25,000,000. To determine the efficacy of any compression algorithm, there are two approaches, the compression ratio, and the space savings. The compression ratio is calculated as follows:

$$Compression\ Ratio = \frac{(Packet\ Data + Header)}{(GNCS\ Data + Header)}$$

The space savings is calculated as follows, as a percentage:

$$Space\ Savings = 1 - \frac{(GNCS\ Data + Header)}{(Packet\ Data + Header)} * 100$$

Appendix F provides the detailed analysis of the two datasets. As has been discussed, the GNCS algorithm is data-driven, in that the more data that is available, the higher the likelihood of finding similarity. To highlight the relationship between the average data per packet for each sample versus the space savings percentage, the data is presented graphically in Figures 5 and 6 for the MACCDC 2012 and UNSW-NB15 17-2-2015 datasets, respectively.

Figure 5 - MACCDC Data Distribution

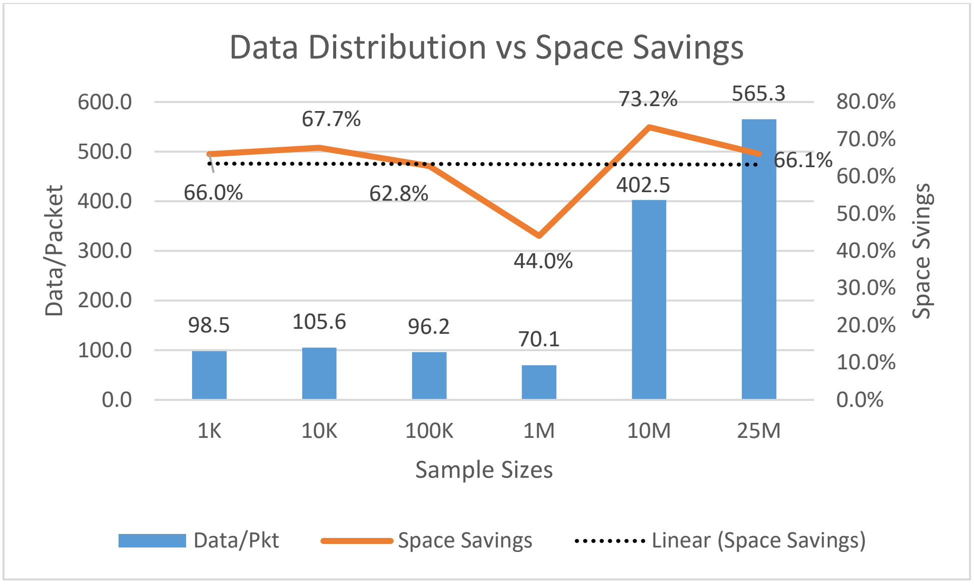
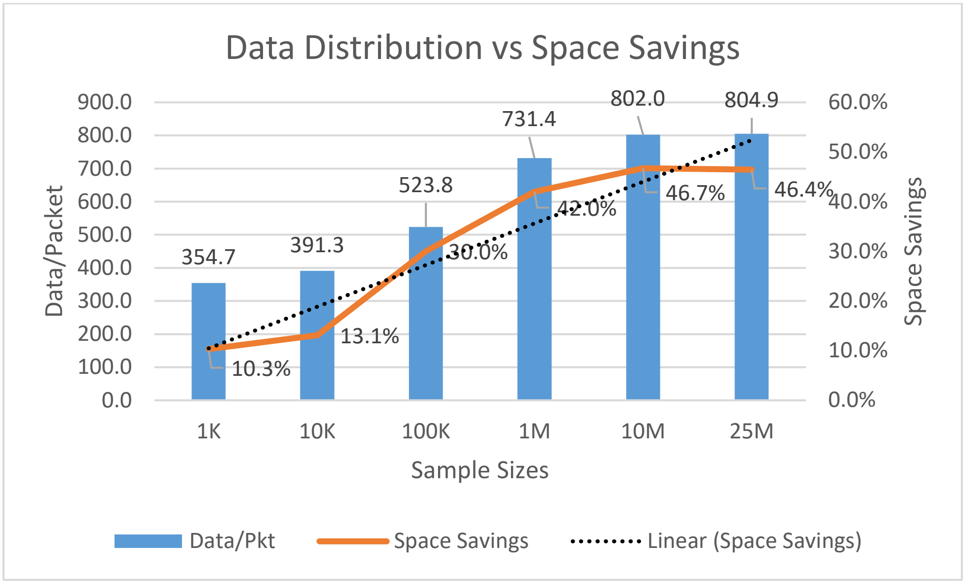


Figure 6 - UNSW-NB15 Data Distribution



Although there are sample-to-sample fluctuations in space savings, the trend line of the MACCDC 2012 dataset is relatively flat at a space savings of approximately 67% (Figure 5), regardless of data volume. However, the UNSW-NB15 17-2-2015 dataset does exhibit an increase in space savings as the packet data increases (Figure 6), with the actual space savings results leveling off at approximately 46%. Therefore, both datasets provide an indication that implementation of genomic compression techniques can have a significant impact on the size of audit/log files consisting of captured Ethernet packets.

Summary

The UNSW-NB15 17-2-2015 dataset compiled by Moustafa and Slay (2015) contains 27 files with a combined 87,492,159 packets. The dataset was analyzed for the occurrence of protocols at the internet layer, yielding just two, the Address Resolution Protocol (ARP) and the Internet Protocol, Version 4 (IPv4). The transport layer exhibited every protocol except one, protocol number 58. The Transmission Control Protocol (TCP) represented 98.342% of the UNSW-NB15 dataset, the User Datagram Protocol (UDP represented 1.635%, and the remaining representing only 0.023% of the total.

The MACCDC 2012 dataset is a network capture of traffic generated in the 2012 Mid-Atlantic Collegiate Cyber Defense Competition where Blue Team students protected a network from a Red Team attack comprised of members of industry and academia. This traffic was interspersed with normal network traffic. The MACCDC 2012 dataset consisted of 17 files with a combined 71,856,691 packets. Unlike the UNSW-NB15 dataset, the MACCDCC dataset consisted of 17 different internet layer protocols. And, although the most-prevalent protocol was IPv4 with 97.07% of the total, IPv6 was second with 1.73% and ARP a distant third with only 0.14%, and the remaining 14 protocols having a combined 1.06%. Unlike the UNSW-NB15 dataset, there were only five transport layer protocols in the MACCDC dataset, with TCP and UDP comprising 98.0% and 0.8%, respectively. The remaining three protocols represented 1.2% of the total.

These two datasets, although comprised of multiple static packet capture files, represented continuous network traffic. The 25-million sample for the UNSW-NB15 dataset required slightly fewer than nine (9) files, and occurred over a period of 3 hours, 48 minutes, 48 seconds, for an average of 549 μs between packets, The 25-million sample for the MACCDC 2012 dataset required slightly over five (5) files, and occurred over a period of 3 hours, 26 minutes, 1 second, for an average of 494 μs between packets.

The genomic compression technique requires packet data to be effective, although the duplicate data elimination component can be applied to message format protocols that have minimal data but are comprised of packets with common header information. Further analysis of the transport layer protocols on both datasets indicate that only the TCP packets contain significant data. Therefore, parser development was limited to parsers for ARP, IPv4, and TCP protocols; the implementation of the actual genomic compression techniques to network traffic was geared solely around TCP.

Although ARP packets are processed, it is solely to eliminate common data, since those packets are too short for the genomic compression techniques. As with ARP, duplicate information in TCP packets is eliminated and just saved once, with the data segments aggregated into packet flows. When the program detected that a flow was complete, the genomic compression technique was then applied in which the flows were analyzed for commonality leading to the creation of the edit scripts. Due to the overhead associated with edit scripts, only flows that exceeded a threshold level were candidates for the techniques.

The results of this research indicated that the application of the genomic compression techniques to network traffic can have a significant impact on audit/log network capture file storage space. Since this research was limited to TCP packets in complete, aggregated network flows, and not the entire file, there was no method available to compare the results to other techniques. Therefore, the only comparison available was between the size of the data processed by the GNCS and the original raw data. The application of genomic compression techniques was shown to save space required for TCP packets – an average of 46% for the UNSW-NB15 dataset and 67% for the MACCDC dataset, as well as an approximate 50% savings for ARP network traffic for either dataset.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Conclusions

The program developed in this research indicates that utilization of genomic compression techniques to network traffic can improve upon standard compression algorithms. A major aspect of the original goal was to develop a compression algorithm that upon decompression, could reconstruct the original packets as closely as possible. Since not every protocol requires every field in their associated packets, it was determined to eliminate superfluous data during processing to save file space. Therefore, this technique is considered a lossy compression technique.

Genomic compression techniques are based on the commonality found in genes. Loh, Baym, and Berger (2012) developed a compression algorithm that took advantage of the common parts of a gene by replacing them with a link and an edit script of the differences. The Genomic Network Compression System (GNCS) presented here utilized that same concept – duplicate packet fields were removed, and common aggregated data flows were replaced with an edit script showing the differences.

Although it was shown that removal of common packet fields in a packet flow could provide a significant reduction if the size of the aggregated packet flows, this is no real improvement on standard packet flow systems (Claise, Trammell, & Aitken, 2013). However, applying the genomic compression techniques to the aggregated data flow provided for a significant reduction in the size of a TCP packet flow – an average of 46% for the UNSW-NB15 17-2-2015 dataset and 67% for the MACCDC 2012 dataset. In addition, ARP network traffic exhibited a savings of approximately 50% for either dataset over the raw PCAP packet data.

The reported space savings are an indication of the capability of genomic compression techniques, and although accurate for the two selected datasets, those datasets are not normal network traffic. The MACCDC dataset is the network packet captures from the 2012 Mid-Atlantic Collegiate Cyber Defense Competition, in which students from different universities act as blue team members to protect a corporate network from attack by a red team made up of volunteers from industry and academia. The network activity of the red and blue teams is interspersed with simulated normal network traffic. As such, there was no indication as to what attack vectors the red team was attempting to exploit, nor how the blue team defended against those attacks. However, this could explain the fluctuations in the GNCS results and the higher space savings compared to the UNSW-NB15 dataset – there could be multiple similar attacks and responses that were easier to match using the algorithm.

The UNSW-NB15 dataset was created by Nour Moustafa and Jill Slay in 2015 to provide a more-current dataset for testing intrusion detection systems. It is a hybrid dataset consisting of both real-world network traffic and synthetic data representing nine types of attacks, namely: Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms. Although Moustafa and Slay (2015) state there is real-world traffic, analysis of the GNCS output identified that 65.75% of all packets in the 25-million packet sample were duplicates (Table 8, highlighted cell). In 2010, Pentikousis et al. performed an analysis on network traces from the National Laboratory of Applied Network Research Passive Measurement and Analysis (NLANR/PMA). This analysis determined that in the various traces in the dataset, retransmissions of TCP packets ranged from 0.48% to 3.6%, with an average of 2.4% (Pentikousis et al., 2010). Moustafa and Slay (2015) provided no explanation for this significant discrepancy, but it is probably safe to assume it was intentional to make the attack packets a smaller percentage of the total and thus harder to identify.

Table 4 – UNSW-NB15 Dataset Output

UNSW-NB15 Dataset – 25-Million Packet Run					
Record Type	Total Records	Total Size	Record Length	Total Packets	% of Total
File header	1	188	188		
TCP complete records	397,262	148,102,391	47	7,496,751	29.99%
TCP orphan records	219,065	15,569,572	48	245,776	0.98%
TCP duplicate records	16,437,802	30,280,976	16	16,437,802	65.75%
TCP ignored records	396,419	3,162,938	Variable	396,419	1.59%
UDP complete records	413,795	5,829,179	Variable	413,795	1.66%
UDP orphan records	0	0	Variable	0	0.00%
ARP complete records	1,561	84,294	54	3,122	0.01%
ARP orphan records	3	132	44	3	0.00%
ARP duplicate records	417	6,672	16	417	0.00%
Unsupported L2 records	0	0	Variable	0	0.00%
Unsupported L3 records	5,915	1,381,679	Variable	5,915	0.02%
Unsupported L4 records	0	0	Variable	0	0.00%
Data records	367,612	6,032,954,363	Variable		
Header records	8	682	Variable		
Totals:	**18,239,860**	**6,237,373,066**		**25,000,000**	**100.0%**

A close examination of the datasets indicated that in some cases, data flows that were exact matches were either nonsense text or encrypted – the protocol did not indicate the traffic was encrypted. This adds credence to the statement that the datasets were artificially expanded. However, in the real world, this would not be observed for different IP addresses, and encrypted data flows would defeat the comparison routines required by the genomic compression algorithm. Privacy and confidentially considerations aside, if the algorithm could be implemented in conjunction with a Transport Layer Security Inspection system (NSA, 2019), more flows would be available for comparison, and the likelihood of finding matches, even partial matches would increase, leading to a higher compression ratio.

One issue with the development was the utilization of an implementation of the Levenshtein algorithm (Levenshtein, 1966) available from the Edpresso Team (Edpresso, 2022). Although this variation is a simple implementation of the Levenshtein algorithm, it requires a single 2-dimensional matrix which, for strings m and n, would be [m + 1] by [n + 1] in size. It is not uncommon that a user download could be up to hundreds of megabytes in size. If you are determining if a one-megabyte string is a substring of a very large download, the matrix could actually be terabytes in size, easily making in unmanageable to process in most computers. For this research, only matrices that would fit in available memory were utilized – if the matrix was too large, the processing of that combination was ignored. Although the extremely large matrices would remain unmanageable, a variation of the Levenshtein algorithm was presented that would break the matrix into multiple, smaller matrices (Hirschberg, 1975). To compare large strings, a variation of the Hirshberg algorithm would need to be implemented.

Implications

An approximately 50% decrease in file size indicates that application of edit scripts similar to genomic compression techniques could be beneficial in storing network traffic. The key for a practical use of the algorithm would be to optimize the exiting parsers and add additional parsers as necessary, such that only the information that is germane to the long-term use is maintained. Although the final step in the current implementation was to combine all the associated interim files into a single, compressible file, use of compressed bitmap indexes could allow for real-time comparisons required for detection of such activity as advanced persistent threats or data exfiltration. Future research would be toward improving efficiency of the approximate match search algorithms. Besides extending the Hirschberg algorithm to a larger number of smaller problems, the use of multi-processing, multi-threading could decrease processing time. This will not be a total solution to the large matrix issue, since even if a matrix is broken into smaller matrices, the total of the computer memory for all the matrices could still exceed available memory. Rather, the smaller matrices would need to be performed sequentially, potential edit strings created (containing the '=', '!', '-' and '+' symbols), and the memory released for use by a subsequent matrix. At the end, the pieces of edit strings would be reconstructed to provide the final string required to create the actual edit script.

For the genomic techniques to be maximized, there needs to be a higher probability of packet data segments containing similar data. This would be the case on a corporate internal network, where users are routinely accessing similar data, such as would be experienced accessing the organizations website, SharePoint sites, etc. This would be even more likely given the larger number of people working from home and accessing the corporate via secure VPN tunnels. Since all data required by a user would be traversing the internet, the probability of common data being shared by multiple users appearing in network audit log files would be increased. Application of genomic compression techniques to potentially a large amount of common data should greatly increase the space savings above that shown in this research. If Transport Layer Security Inspection system (NSA, 2019) is included, there could be substantial decreases in space required to save audit logs.

Recommendations

Future research would further refine the algorithm based upon the goal of the process. Implementation for identifying APTs might be different than an implementation to detect insider threats. The initial approach was geared totally to decreasing file size, based on aggregated data segment flows. A simple extension would be to maintain the 4-tuple of source and destination IP addresses and ports and a link to the data flows in a database. This way, trends in usage related to specific IP addresses could be identified as they occur. Conversely, if data exfiltration is suspected, it would be possible to trace back to the user or users involved in the activity.

Without regard to possible end uses, the main thrust of future research will be toward improving the efficiency of the approximate match algorithm. As discussed above, there are several published extensions to the Levenshtein and Hirschberg algorithms. During the research, a new approach was identified in which creation of the symbolic edit strings is concurrent with the comparison of two strings, rather than creating the large matrix first, and then reading that matrix backwards to develop the symbolic edit strings. This would not only decrease the memory requirement of the current approach but would also be a candidate for multi-processing – as possible common beginnings are identified, a separate process is spawned to continue that analysis, while the original process continues looking for additional possible matches.

Conceptually, in a production environment, as packets are captured, they would be processed, and results stored for future analysis. This research utilized balanced binary search trees to maintain the incomplete flows and ARP pairs. In production, these would be replaced with a database that would maintain references to all flows, such that queries could be made to perform trend analysis on any combination of IP addresses and ports. Considering the potential file size of one year of audit logs, it would not be practical to maintain all that data in a live environment. In the tests of the MACCDC and UNSW-NB15 datasets, the data flows made up 83.2% and 96.7% of the total output data files, respectively. Further research will be required to determine the most-efficient method of storing the data off-line but make it available for research.

Summary

While business traffic has doubled every three years since 2016 (O'Dea, 2020), the cost of cybercrime has kept pace (Embroker Team, 2021). To combat cybercrime, organization began collecting and maintaining audit files to aid in identification of malicious activity (Marker, 2021; Shopp, 2020; NIST ITL Bulletin, 1997). In 2015, Horne presented data traffic from HP DNS servers. Assuming the DNS request is a prelude to actual connections, the data can be extrapolated to possible IPv4 traffic. Using packet estimates provided by Kim, et, al., (2004), an organization with 1,000 users could generate in excess of 10 petabytes of audit logs per year.

To address the storage issues of such large files, various compression algorithms have been developed, such as ZIP, RAR, and TAR, although these are generally used for archiving purposes (FileInfo, n.d.). There are two general categories of compression algorithms: Lossless and Lossy (Kavitha, 2016). Lossless compression is used where exact copies of the original data is required, such as with run length encoding, Lempel Ziv Welch and Huffman encoding; Lossy compression allows for elimination of some of the original data that is deemed not to impact the essential information of the original, such as JPEG, MP3, and MP4 (Kavitha, 2016).

Genomic research involves analysis of very large datasets. In 2012, Loh, Baym and Berger proposed a compression algorithm that allowed them to perform genomic analysis directly on the compressed data, without having to be uncompressed. The purpose of this research is to utilize the dynamic compression techniques of Loh, et al. (2012) to not only reduce the size of the network data file storage, but also allow analysis in real time. The compression algorithm developed here is considered lossy, since certain packet fields are eliminated during processing.

The development and validation of the genomic compression technique required the availability of a dataset that was representative of the real world. After an in-depth search, the UNSW-NB15 17-2-2015 dataset compiled by Moustafa and Slay (2015) and the network packet captures from the 2012 Mid-Atlantic Collegiate Cyber Defense Competition were selected. These files follow the TCP/IP, 4-layer model, but utilized different PCAP capture formats, requiring modification of the program to account for the network layer header differences.

The datasets were analyzed for the occurrence of protocols at the internet and transport layers. At the internet layer, there are only two protocols represented in the UNSW-NB15 dataset, the Address Resolution Protocol (ARP) and the Internet Protocol, Version 4 (IPv4). However, in addition to those two protocols, the MACCDC dataset contained 12 additional protocols, none of which was of significant occurrence to warrant further analysis. With both datasets, the Transmission Control Protocol (TCP) was the most-prevalent protocol in the transport layer, representing 98.3% and 90.7% of the UNSW-NB15 and MACCDC datasets, respectively. However, after TCP, the remaining transport layer protocols varied between the two datasets. The User Datagram Protocol (UDP) was the next for both, but there was no subsequent similarity between the two datasets. The MACCDC dataset had a total of five transport layer protocols, whereas the UNSW-NB15 dataset exhibited every possible protocol, except for protocol number 58; 239 of which only had 12 instances each. Due to the minimal occurrences of these protocols and their lack of significant data, all were ignored for further processing. Ultimately, modules were developed to parse the datasets for ARP and IPv4 packets, and the latter further parsed into just TCP and UDP usage – all other transport layer protocols were saved as is, without taking advantage of the genomic techniques. As development progressed, it was apparent that the bulk of the UDP usage was for single-packet messaging. Therefore, as with unsupported protocols, all UDP packets were saved as is.

The program developed for this research read through the files of the datasets, sequentially, packet by packet. Since only ARP and TCP packets were matched, they were saved in their own balanced binary tree structure. All other packets were immediately written to their associated interim file: 1) TCP duplicate packets; 2) UDP packets; 3) ARP duplicate packets; 4) Ignored TCP packets; 5) Unsupported Internet Layer packets; 6) Unsupported Transport Layer packets; and 7) Unsupported TCP protocols packets.

The Address Resolution Protocol is a fixed-length protocol, consisting of request and reply packets. Although there is no data segment and thus no ability to apply the genomic compression technique, by combining the request and reply pair into a single, final structure, significant space can be saved. When an ARP request packet is read, it is written to the ARP binary tree. Upon identification of the reply packet, it is matched with the request packet, the final structure is written to its own interim file, and the associated request node removed from the ARP binary tree. Each ARP packet is 28 bytes in length (Figure 2, Chapter 4), for a combined 56 bytes per pair. However, in a PCAP file, each packet has a 16-byte header, making the total 88 bytes for the pair. The final structure for an ARP pair utilized in this research is 46 bytes, for a 48% savings in file space.

As indicated above, the implementation of the genomic compression techniques to network traffic was geared solely around TCP packets. To decrease any overhead associated with creating edit scripts, it was decided that complete packet flows would be aggregated, rather than attempting to apply the genomic techniques to individual packet data segments. Since the protocol is known (TCP), the aggregation was based on a 4-tuple key, comprised of the source and destination IP addresses and ports, as opposed to the 5-tuple key utilized in the IP Flow export protocol, which includes the protocol in the key (Claise et. al., 2013). This had the added advantage of eliminating the common data exhibited in the packets that make up packet flow – the important information for reconstruction was save once, with a link to the data. As packets are read, the first packet with a unique 4-tuple key is written to the TCP balanced binary tree, and the data segment stored in an associated linked list, ordered by sequence number. As additional packets matching the 4-tuple key are identified. If they are duplicates, they are immediately written to the TCP duplicate interim file. If not, the data segment is added to the sequence linked list. This processing of the flow continues until the last packet in the flow is received (FIN, or finish, flag is set). When the program detected that a flow was complete, the genomic compression technique was then applied. Since the creation of edit scripts requires overhead to identify the components of the script, only flows that exceed a threshold level were possible candidates for the techniques.

The major difference between actual genomic data and network data is that with gene research, a single gene is compared to a known “dictionary” and the difference between the two are identified. With network traffic, there is no one-to-one relationship. Rather, the current packet flow must be compared to every previous packet flow that has been saved, a one-to-many relationship. Besides ensuring the packet flow meets a minimum size threshold, determining the longest common substring between segments of the packet flow and like-length segments of the previous flows was utilized to identify possible common strings. It was at this point that the Levenshtein Edit Distance algorithm (Levenshtein, 1966) was applied to the two flows (a known flow and the test flow) and the resultant matrix walked in reverse to determine where characters either matched, did not match, and whether the insertion or deletion of a character to either string would be required. The edit string was comprised of four symbols: ‘=’ where the characters matched; ‘!’ where the character did not match; ‘+’ where a character needed to be inserted into the known string; and ‘-’ where a character had to be removed from the known string. This edit string was then used to create the edit scripts to indicate how to change a known flow to match the current flow.

There is a single edit script node structure comprised of four fields: 1) the edit type, which is either ‘K’ for a reference to the known string, or ‘T’ referring to the test string; 2) the starting location for either string; 3) the length of the data; and 4) the data involved. In the case of the node referring to a known string, the data field is empty since the known string is on file and the location and length of the common data has been saved in the previous two fields. In the case of the test string, the data field would contain the unique data segment belonging to the test string. At this point, the data flow, if unique or didn’t meet the length threshold, is written to the data interim file. If a match is identified, the edit scripts are written to the datafile. In either case, the base TCP packet is removed from the TCP binary tree, including removal of the sequence linked list.

At the end of processing of each dataset, the program creates the final file, based on the various interim files, as well as any orphan nodes in the ARP and TCP binary trees. The final file is built upon the TCP interim file. During its initial creation at program start-up, the first 188 bytes are written with zeros, as reserved space for the final file header. Final processing is performed in the following order, based upon the header structure: 1) During processing, the count of the number of flows was stored, so the TCP flow record location, record count, and the known format length are written to the header; 2) the TCP binary tree is checked for any orphan nodes. Their starting location in the file is saved and the orphan packets were written to file. The header information is updated and the remaining TCP binary tree nodes removed and the tree destroyed; 3) the duplicate TCP interim file is copied to the final file and the header information updated; 4) since UDP packets were not processed for flows, there is nothing to add to the file and the header is updated for location with a count of zero records; 5) all UDP packets were written to an interim file, and considered orphans. The UDP interim file was copied to the final file and the header information updated; 6) the ARP completed pair interim file is copied to the final file and the header information updated; 7) the ARP binary tree is checked for any orphans and the associated packets are written to the output file, the header information updated; and the ARP binary tree deleted. 8) the ignored TCP, unsupported Internet Layer protocol, unsupported Transport Layer, and unsupported TCP interim files are copied to the final file and the header information updated, in the order presented; 9) the data interim file is copied to the output file and the header information updated; and 10) the number of files in the dataset is written, followed by the header information for each file, and then the location of the dataset header files is written to the final file header.

The results of this research indicated that the application of the genomic compression techniques to network traffic can have a significant impact on audit/log network capture file storage space. For protocols with significant data, the application of genomic compression techniques was shown to save significant space – an average of 46% for the UNSW-NB15 dataset and 67% for the MACCDC dataset for TCP aggregated data flows. For message formats consisting of request packets and replies, the genomic compression technique of eliminating duplicate data can also provide significant space savings – both datasets exhibited approximately 50% space savings.

Appendix A

NIST SP 800-53 Revision 5, Audit and Accountability Security Controls

Table A1 provides a list of the NIST SP 800-53 security and privacy controls and control enhancements that are assigned to the Audit and Accountability family. Note that the breaks in numbering are due to controls from the previous Revision 4 being withdrawn and incorporated into other controls. For simplicity, those controls have been eliminated from this table. The controls that are pertinent to this research are highlighted in yellow.

Table A1 - NIST SP 800-53 Revision 5 Audit and Accountability Security Controls

Control Number	Control Name
AU-01	Policy and Procedures
AU-02	Event Logging
AU-03	Content of Audit Records
AU-03(1)	Content of Audit Records | Additional Audit Information
AU-03(3)	Content of Audit Records | Limit Personally Identifiable Information Elements
AU-04	Audit Log Storage Capacity
AU-04(1)	Audit Log Storage Capacity | Transfer to Alternate Storage
AU-05	Response to Audit Logging Process Failures
AU-05(1)	Response to Audit Logging Process Failures | Storage Capacity Warning
AU-05(2)	Response to Audit Logging Process Failures | Real-time Alerts
AU-05(3)	Response to Audit Logging Process Failures | Configurable Traffic Volume Thresholds
AU-05(4)	Response to Audit Logging Process Failures | Shutdown on Failure
AU-05(5)	Response to Audit Logging Process Failures | Alternate Audit Logging Capability
AU-06	Audit Record Review, Analysis, and Reporting
AU-06(01)	Audit Record Review, Analysis, and Reporting | Automated Process Integration

Control Number	Control Name
AU-06(03)	Audit Record Review, Analysis, and Reporting | Correlate Audit Record Repositories
AU-06(04)	Audit Record Review, Analysis, and Reporting | Central Review and Analysis
AU-06(05)	Audit Record Review, Analysis, and Reporting | Integrated Analysis of Audit Records
AU-06(06)	Audit Record Review, Analysis, and Reporting | Correlation with Physical Monitoring
AU-06(07)	Audit Record Review, Analysis, and Reporting | Permitted Actions
AU-06(08)	Audit Record Review, Analysis, and Reporting | Full Text Analysis of Privileged Commands
AU-06(09)	Audit Record Review, Analysis, and Reporting | Correlation with Information from Nontechnical Sources
AU-07	Audit Record Reduction and Report Generation
AU-07(1)	Audit Record Reduction and Report Generation | Automatic Processing
AU-08	Time Stamps
AU-09	Protection of Audit Information
AU-09(1)	Protection of Audit Information | Hardware Write-once Media
AU-09(2)	Protection of Audit Information | Store on Separate Physical Systems or Components
AU-09(3)	Protection of Audit Information | Cryptographic Protection
AU-09(4)	Protection of Audit Information | Access by Subset of Privileged Users
AU-09(5)	Protection of Audit Information | Dual Authorization
AU-09(6)	Protection of Audit Information | Read-only Access
AU-09(7)	Protection of Audit Information | Store on Component with Different Operating System
AU-10	Non-repudiation
AU-10(1)	Non-repudiation | Association of Identities
AU-10(2)	Non-repudiation | Validate Binding of Information Producer Identity
AU-10(3)	Non-repudiation | Chain of Custody
AU-10(4)	Non-repudiation | Validate Binding of Information Reviewer Identity
AU-11	Audit Record Retention
AU-11(1)	Audit Record Retention | Long-term Retrieval Capability
AU-12	Audit Record Generation
AU-12(1)	Audit Record Generation | System-wide and Time-correlated Audit Trail
AU-12(2)	Audit Record Generation | Standardized Formats
AU-12(3)	Audit Record Generation | Changes by Authorized Individuals
AU-12(4)	Audit Record Generation | Query Parameter Audits of Personally Identifiable Information
AU-13	Monitoring for Information Disclosure
AU-13(1)	Monitoring for Information Disclosure | Use of Automated Tools
AU-13(2)	Monitoring for Information Disclosure | Review of Monitored Sites

Control Number	**Control Name**
AU-13(3)	Monitoring for Information Disclosure | Unauthorized Replication of Information
AU-14	Session Audit
AU-14(1)	Session Audit | System Start-up
AU-14(3)	Session Audit | Remote Viewing and Listening
AU-16	Cross-organizational Audit Logging
AU-16(1)	Cross-organizational Audit Logging | Identity Preservation
AU-16(2)	Cross-organizational Audit Logging | Sharing of Audit Information
AU-16(3)	Cross-organizational Audit Logging | Disassociability

Appendix B

Dataset Analysis and Selection

Table B1 provides the complete list of the downloaded candidate datasets. Both the MACCDC and UNSW-NB15 datasets were single datasets comprised of multiple, chronologically sequential PCAP files. However, the two MILCOM 2016 Datasets were actually comprised of multiple, individual datasets

Table B1 – Candidate Datasets

Mid-Atlantic Collegiate Cyber Defense Competition (MACCDC) Datasets			
Dataset Name	Compressed Size	#PCAP Files	#Datasests
MACCDC 2010	10 GB	27	1
MACCDC 2011	14.2 GB	15	1
MACCDC 2012	5.4 GB	17	1
University of New South Wales (UNSW) Datasets			
Dataset Name	Compressed Size	#PCAP Files	#Datasests
UNSW-NB15 17-2-2015	49 GB	27	1
UNSW-NB15 22-1-2015	50.2 GB	53	1
Military Communications Conference (MILCOM) 2016 Datasets			
Dataset Name	Compressed Size	#PCAP Files	#Datasests
dataSetAv2	33 G	15	15
dataSetEAggregator	44 G	48	48

Each of the candidate datasets was processed by various modules of the utility program which incorporated certain built-in functions of the Npcap library. In the PCAP file format, each packet is preceded with a packet header as follows:

```
typedef struct pcap_record_header {
    unsigned int32        timestamp_seconds;          // 4 bytes
    unsigned int32        timestamp_microseconds;     // 4 bytes
    unsigned int32        packet_octets_captured;     // 4 bytes
    unsigned int32        actual_packet_length;       // 4 bytes
}                                                     // 16 total bytes
```

The first module read through all files of the dataset, totaling the number of PCAP records and two values provided by the PCAP packet header: the number of bytes captured (packet_octets_captured) and saved in the PCAP file; and the length, in bytes, of the record actually on the network when it was captured (actual_packet_length). To provide a comparative analysis, the average packet size was calculated for each dataset. After counting all the packets and bytes in a file, the module then calculated the expected size of the PCAP file to account for the PCAP packet header (16 bytes per packet) and the global PCAP file header (24 bytes per file). The last value obtained was the actual on-disk file size. Tables B2, B3, B4, and B5 show the summary results for the MACCDC, UNSW-NB15, MILCOM dataSetAv2, and MILCOM dataSetEAggregator datasets, respectively (Appendix C, Tables C1 through C3 provides the per-file results for the three MACCDC datasets and Appendix D, Tables D1 and D2 for the two UNSW-NB15 datasets). Putting the average size into perspective, any comparisons must account for the impact of the lengths of the various protocol packet headers. The network layer header ranges from 14 bytes to 18 bytes and the Internet layer (IPv4) header is 20 bytes. Transport layer headers are variable. For example, TCP headers are 20 bytes, while UDP headers are 8 bytes. Therefore, mosts packet will require anywhere from 40 to 60 bytes for the headers, which must be included in any data segment calculation.

Table B2 – Candidate Datasets: File and Packet Statistics – MACCDC Datasets

MACCDC Datasets			
Dataset:	**MACCDC 2010**	**MACCDC 2011**	**MACCDC 2012**
Total Packets	264,973,151	134,465,786	71,856,691
Total Bytes	32,513,443,821	30,237,514,309	16,703,208,838
Bytes/Packet	123	225	222
Bytes on Wire	32,513,443,821	30,237,514,309	16,703,208,838
Total w/Headers	36,753,014,885	32,388,967,245	17,852,916,302
Actual File Size	36,753,014,885	32,388,967,245	17,852,916,302

The MACCDC datasets results did not show anything abnormal, although the average packet size of the MACCDC 2010 dataset was approximately half the sizes of the MACCDC 2011 and MACCDC 2012 datasets (Table B2). Even though that dataset contained significantly more packets than the other two, due to the lower data segment sizes, the MACCDC 2010 dataset was eliminated from further analysis (dataset name highlighted in red in Table B2).

Table B3 – Candidate Datasets: File and Packet Statistics – UNSW-NB15 Datasets

UNSW-NB15 Datasets		
Dataset:	**17-2-2015**	**22-1-2015**
Total Packets	87,492,159	94,571,342
Total Bytes	48,347,208,654	51,116,871,461
Average Bytes/Pkt	411	96
Bytes on Wire	48,347,208,654	51,116,871,461
Total w/Headers	49,747,083,846	52,630,014,205
Actual File Size	49,747,092,020	53,044,845,688

In both UNSW-NB15 datasets, the actual file sizes where larger than the calculated size for the number of packets read (highlighted in yellow in Table B3). For the UNSW-NB 17-2-2015 dataset, the discrepancy was minor: 8,174 additional bytes in a dataset that captured 49,747,083,846 bytes. With the UNSW-NB 22-1-2015 dataset, there was a more significant difference: 414,831,483 bytes compared to 52,630,014,205 captured bytes. In both datasets, the extra data did not cause any program issues and the packet counts were confirmed using Wireshark. However, due to the significantly smaller packet sizes in the 22-1-2015 dataset (96 bytes per packet versus 411 bytes per packet for the 17-2-2015 dataset), the 22-1-2015 dataset was excluded from further analysis (dataset name highlighted in red in Table B3).

As with the UNSW-NB15 datasets, two of the MILCOM 2016 dataSetAv2 datasets contained additional data (cells highlighted in yellow in Table B4). However, the differences were very significant with the actual file size of the i-dc-9 dataset being over twice what was expected and that for the fw-sniffer dataset being 9.1 times larger. Again, these differences did not impact processing and the packet counts were confirmed with Wireshark. However, like the UNSW-NB15 22-1-2015 dataset, the fw-sniffer dataset only contained 96 bytes per packet, so it was excluded from future processing, as well as those candidate datasets with less than 1,000,000 packets (dataset names highlighted in red in Table B4).

The other MILCOM 2016 dataset group, dataSetEAggregator, also had files with extra data, and similar to the dataSetAv2 group, the differences were significant: the cop dataset was 10.9 times larger than expected and the i-dc-9 dataset was 5.6 times larger (highlighted in yellow in Table B5). Again, the extra data did not impact processing and the number of packets was confirmed with Wireshark. Since the cop dataset packets were significantly less than other datasets with greater than 1,000,000 packets, it was excluded from further analysis, as were the two datasets with no packets (1stplt2rto and 2ndpltrto), and the other datasets that did not reach the 1,000,000-packet threshold (dataset names highlighted in red in Table B5). This eliminated every dataset in the dataSetEAggregator group, except the i-dc-9 dataset.

After the first level of analysis, most of the datasets have been eliminated either due to number of packets to test, or the average packet not containing sufficient data to test. This leaves eight datasets for further analysis: MACCDC 2011; MACCDC 2012; UNSW-NB15 17-2-2015; the i-dc-4, i-dc-5, i-dc-6, and i-dc-9 datasets from the dataSetAv2 group; and the i_dc-9 dataset from the dataSetEAggregator group.

Table B4 – Candidate Datasets: File and Packet Statistics – MILCOM Datasets

MILCOM 2016 dataSetAv2 Datasets					
Dataset:	db	fw-sniffer	i-dc-1	i-dc-2	i-dc-3
Total Packets	278,827	14,239,985	59,443	15,706	77,804
Total Bytes	102,331,494	1,368,367,400	52,124,194	6,175,912	67,893,178
Average Bytes/Pkt	411	96	877	393	873
Bytes on Wire	102,331,494	1,368,367,400	52,124,194	6,175,912	67,893,178
Total w/Headers	106,792,750	1,596,207,184	53,075,306	6,427,232	69,138,066
Actual File Size	106,792,750	14,481,109,072	53,075,306	6,427,232	69,138,066
Dataset:	i-dc-4	i-dc-5	i-dc-6	i-dc-7	i-dc-8
Total Packets	1,580,379	2,124,915	1,428,356	26,319	461,571
Total Bytes	1,622,623,997	2,172,641,100	1,464,901,945	11,792,338	459,676,581
Average Bytes/Pkt	1,027	1,022	1,026	448	996
Bytes on Wire	1,622,623,997	2,172,641,100	1,464,901,945	11,792,338	459,676,581
Total w/Headers	1,647,910,085	2,206,639,764	1,487,755,665	12,213,466	467,061,741
Actual File Size	1,647,910,085	2,206,639,764	1,487,755,665	12,213,466	467,061,741
Dataset:	i-dc-9	nodejs	ns-dc	smf	wpvuln
Total Packets	7,487,262	18	318,341	432,053	7,414
Total Bytes	3,415,530,780	4,743	28,359,627	154,175,229	1,381,336
Average Bytes/Pkt	456	264	89	357	186
Bytes on Wire	3,415,530,780	4,743	28,359,627	154,175,229	1,381,336
Total w/Headers	3,535,326,996	5,055	33,453,107	161,088,101	1,499,984
Actual File Size	7,830,294,292	5,055	33,453,107	161,088,101	1,499,984

Table B5 – Candidate Datasets: File and Packet Statistics – MILCOM Datasets

MILCOM 2016 dataSetEAggregator Datasets					
Dataset:	1stplt1rto	1stplt1sqd1rflmn1	1stplt1sqd1rflmn2	1stplt1sqd1tl	1stplt1sqd2rflmn1
Total Packets	3,069	11,264	9,423	10,417	9,126
Total Bytes	3,264,894	5,944,570	3,957,648	4,749,122	3,737,216
Avg Bytes/Pkt	1,064	528	420	456	410
Bytes on Wire	3,264,894	5,944,570	3,957,648	4,749,122	3,737,216
Total w/Headers	3,314,022	6,124,818	4,108,440	4,915,818	3,883,256
Actual File Size	3,314,022	6,124,818	4,108,440	4,915,818	3,883,256
Dataset:	1stplt1sqd2rflmn2	1stplt1sqd2tl	1stplt1sqdldr	1stplt2rto	1stplt2sqd1rflmn1
Total Packets	9,063	10,226	9,818	0	9,125
Total Bytes	3,556,794	4,673,850	4,334,082	0	3,552,190
Avg Bytes/Pkt	392	457	441	0	389
Bytes on Wire	3,556,794	4,673,850	4,334,082	0	3,552,190
Total w/Headers	3,701,826	4,837,490	4,491,194	0	3,698,214
Actual File Size	3,701,826	4,837,490	4,491,194	0	3,698,214
Dataset:	1stplt2sqd1rflmn2	1stplt2sqd2tl	1stplt2sqd2rflmn1	1stplt2sqd2rflmn2	1stplt2sqd2tl
Total Packets	8,760	9,234	7,867	10,140	9,234
Total Bytes	3,182,094	3,654,292	2,349,358	4,541,954	3,654,292
Avg Bytes/Pkt	363	396	299	448	396
Bytes on Wire	3,182,094	3,654,292	2,349,358	4,541,954	3,654,292
Total w/Headers	3,322,278	3,802,060	2,475,254	4,704,218	3,802,060
Actual File Size	3,322,278	3,802,060	2,475,254	4,704,218	3,802,060
Dataset:	1stplt2sqdldr	1stpltldr	2ndplt1sqd1rflmn1	2ndplt2sqd1rflmn2	2ndplt1sqd1tl
Total Packets	9,522	23,371	11,095	8,347	10,569
Total Bytes	4,011,744	18,919,020	5,844,668	2,856,298	4,998,112
Average Bytes/Pkt	421	810	527	342	473
Bytes on Wire	4,011,744	18,919,020	5,844,668	2,856,298	4,998,112
Total w/Headers	4,164,120	19,292,980	6,022,212	2,989,874	5,167,240
Actual File Size	4,164,120	19,292,980	6,022,212	2,989,874	5,167,240
Dataset:	2ndplt1sqd2rflmn1	2ndplt1sqd2rflmn2	2ndplt1sqd2tl	2ndplt1sqdldr	2ndplt2sqd1rflmn1
Total Packets	8,425	8,569	9,849	12,587	8,456
Total Bytes	3,051,478	3,134,318	4,357,484	7,380,692	2,970,188
Average Bytes/Pkt	362	366	442	586	351
Bytes on Wire	3,051,478	3,134,318	4,357,484	7,380,692	2,970,188
Total w/Headers	3,186,302	3,271,446	4,515,092	7,582,108	3,105,508
Actual File Size	3,186,302	3,271,446	4,515,092	7,582,108	3,105,508

MILCOM 2016 dataSetEAggregator Datasets					
Dataset:	2ndplt2sqd1rflmn2	2ndplt2sqd1tl	2ndplt2sqd2rflmn1	2ndplt2sqd2rflmn2	2ndplt2sqd2tl
Total Packets	8,347	9,300	7,810	10,015	9,522
Total Bytes	2,856,298	3,931,340	2,362,258	4,436,966	3,984,258
Average Bytes/Pkt	342	423	302	443	418
Bytes on Wire	2,856,298	3,931,340	2,362,258	4,436,966	3,984,258
Total w/Headers	2,989,874	4,080,164	2,487,242	4,597,230	4,136,634
Actual File Size	2,989,874	4,080,164	2,487,242	4,597,230	4,136,634
Dataset:	2ndplt2sqdldr	2ndpltldr	2ndpltrto	3rdplt1sqd1rflmn1	3rdplt1sqd1rflmn2
Total Packets	9,698	10,829	0	8,594	8,676
Total Bytes	4,286,814	5,578,678	0	3,132,412	3,203,772
Average Bytes/Pkt	442	516	0	364	369
Bytes on Wire	4,286,814	5,578,678	0	3,132,412	3,203,772
Total w/Headers	4,442,006	5,751,966	0	3,269,940	3,342,612
Actual File Size	4,442,006	5,751,966	0	3,269,940	3,342,612
Dataset:	3rdplt1sqd1tl	3rdplt1sqd2rflmn1	3rdplt1sqd2rflmn2	3rdplt1sqd2tl	3rdplt1sqdldr
Total Packets	9,951	8,802	8,708	10,444	12,847
Total Bytes	4,261,960	3,387,814	3,223,284	5,043,220	7,612,416
Average Bytes/Pkt	428	385	370	483	593
Bytes on Wire	4,261,960	3,387,814	3,223,284	5,043,220	7,612,416
Total w/Headers	4,421,200	3,528,670	3,362,636	5,210,348	7,817,992
Actual File Size	4,421,200	3,528,670	3,362,636	5,210,348	7,817,992
Dataset:	3rdpltldr	3rdpltrto	cop	db	i-dc-9
Total Packets	16,228	7,152	12,974,530	46,831	14,842,234
Total Bytes	11,254,030	7,567,192	1,529,379,506	17,819,978	3,535,380,449
Avg Bytes/Pkt	693	1,058	118	381	238
Bytes on Wire	11,254,030	7,567,192	1,529,379,506	17,819,978	3,535,380,449
Total w/Headers	11,513,702	7,681,648	1,736,972,010	18,569,298	3,772,856,217
Actual File Size	11,513,702	7,681,648	18,916,841,194	18,569,298	20,952,725,401
Dataset:	nodejs	ns-dc	Smf		
Total Packets	452	82,370	73,689		
Total Bytes	30,902	7,441,446	26,756,753		
Average Bytes/Pkt	68	90	363		
Bytes on Wire	30,902	7,441,446	26,756,753		
Total w/Headers	38,158	8,759,390	27,935,801		
Actual File Size	38,158	8,759,390	27,935,801		

In order to provide further analysis of the suitability of any given dataset, the actual structure of the packet captures had to be determined. There are multiple possible packet capture formats within the Npcap library. The differences are related to how the Network Layer header is presented, including whether the PCAP packet starts immediately with the

Internet Layer, totally ignoring the Network Layer header (Tcpdump, 2022). The complete global PCAP file header is as follows:

```
typedef struct pcap_global_header {
    unsigned int32          magic_number;             // 4 bytes
    unsigned int16          version_major;            // 2 bytes
    unsigned int16          version_minor;            // 2 bytes
             int32          time_zone_correction;     // 4 bytes
    unsigned int32          timestamp_accuracy;       // 4 bytes
    unsigned int32          max_capture_length;       // 4 bytes
    unsigned int32          data_link_type;           // 4 bytes
}                                                     // 24 total bytes
```

The "data_link_type" member of the PCAP file header structure represents the Network Layer type (Wireshark, 2020). A "magic_number" structure element is used to identify whether the PCAP file data elements have been written in the native byte order of the current operating system or not. The result of this element determined how the actual data values are extracted from the packet. The next program module to run simply opened the PCAP file using the Ncpap library, displayed the contents of the PCAP file header and closed the file. The Network Layer Type and Native Byte Order for the remaining eight datasets are shown in Table B6.

This module showed that the UNSW-NB15 dataset utilizes network layer header type 113, an artificial network layer header inserted by file captures performed on a Linux system. The two MACCDC and the MILCOM 2016 datasets all utilize network layer header type 1 which references the standard Ethernet link-layer (Network Layer) header. All further analysis, including the actual genomic compression program, utilized a short function to identify the network layer type and data order of the PCAP file, in order to properly account for the length and content of the Network Layer header and obtain accurate data extraction.

Table B6 – Candidate Datasets: File Header Extracts

Dataset	Network Layer type	Native Byte Order
MACCDC 2011	1 (Ethernet)	Native
MACCDC 2012	1 (Ethernet)	Native
UNSW-NB15 17-2-2015	113 (Linux "cooked")	Native
dataSetAv2 i-dc-4	1 (Ethernet)	Native
dataSetAv2 i-dc-5	1 (Ethernet)	Native
dataSetAv2 i-dc-6	1 (Ethernet)	Native
dataSetAv2 i-dc-9	1 (Ethernet)	Native
dataSetEAggregator i_dc-9	1 (Ethernet)	Native

The next analysis read through the datasets, extracting the Internet Layer protocol from the Network Layer header based on the structure defined by the Network Layer type. This module was limited to counting occurrences of the Internet Layer protocols. The results for the eight candidate datasets are provided in Table B7. Except for the two MACCDC datasets, the other six candidate datasets were comprised of two Internet Layer protocols only: Internet Protocol version 4 (IPv4) and the Address Resolution Protocol (ARP), where IPv4 comprised over 99.9% of the total packets. In addition to IPv4 and ARP, the MACCDC datasets also contained several other protocols, the most notable being IPv6. IPv4 again was the dominant protocol with 97.8% for the MACCDC 2011 dataset and 97.0% for the MACCDC 2012 dataset, with IPv6 comprising 1.5% and 1.8% of the MACCDC 2011 and MACCDC 2012 datasets, respectively. (Appendix C, Tables C4 through C6 provide the per-file results for the most-common Internet Layer protocols for the two remaining MACCDC datasets and Appendix D, Tables D3 and D4, for the two UNSW-NB15 datasets).

Table B7 – Candidate Datasets: Internet Layer Protocols

Dataset	Protocol		Description	Count	Percent
	Decimal	Hex			
MACCDC 2011	2,048	0x0800	IPv4	131,527,680	97.815%
	34,525	0x86DD	IPv6	2,010,213	1.495%
	2,054	0x0806	ARP	169,498	0.126%
	36,864	0x9000	Loopback	16,058	0.012%
	35,020	0x88CC	Link Layer Discovery Protocol (LLDP)	5,194	0.004%
	32,821	0x8035	RARP	485	0.000%
			Remaining 11 Protocols:	736,658	0.548%
			Dataset Total:	134,465,786	100.000%
MACCDC 2012	2048	0x0800	IPv4	69,749,729	97.068%
	34525	0x86DD	IPv6	1,246,065	1.734%
	2054	0x0806	ARP	101,294	0.141%
	24579	0x6003	DEC DECNET Phase IV Route	8,870	0.012%
	36864	0x9000	Loopback	6,688	0.009%
	35020	0x88CC	Link Layer Discovery Protocol (LLDP)	2,160	0.003%
	32821	0x8035	RARP	590	0.001%
			Remaining 9 Protocols:	741,295	1.032%
			Dataset Total:	71,856,691	100.000%
17-2-2015	2048	0x0800	IPv4	87,480,078	99.986%
	2054	0x0806	ARP	12,081	0.014%
			Dataset Total:	87,492,159	100.000%
dataSetAv2 i-dc-4	2048	0x0800	IPv4	1,580,285	99.994%
	2054	0x0806	ARP	94	0.006%
			Dataset Total:	1,580,379	100.000%
dataSetAv2 i-dc-5	2048	0x0800	IPv4	2,124,799	99.995%
	2054	0x0806	ARP	116	0.005%
			Dataset Total:	2,124,915	100.000%
dataSetAv2 i-dc-6	2048	0x0800	IPv4	1,428,078	99.981%
	2054	0x0806	ARP	278	0.019%
			Dataset Total:	1,428,356	100.000%
dataSetAv2 i-dc-9	2048	0x0800	IPv4	7,487,190	99.999%
	2054	0x0806	ARP	72	0.001%
			Dataset Total:	7,487,262	100.000%
dataSetEA i_dc-9	2048	0x0800	IPv4	14,842,071	99.999%
	2054	0x0806	ARP	163	0.001%
			Dataset Total:	14,842,234	100.000%

Although the bytes per packet calculations presented in tables B2 through B5 indicated that there are data segments available beyond the various protocol headers, the next analysis identified whether there was any actual usable data for genomic compression. In this run, the transport layer header was parsed to determine the expected data length, accounting for the lengths of the network, internet, and transport layer headers. The data elements extracted and/or calculated included the minimum, maximum, and average expected and actual data lengths, as well as the captured lengths. Table B8 provides the transport layer results for the candidate datasets (Appendix C, Tables C7 and C8 provides for the per-file results for the top Transport layer protocols for the two remaining MACCDC datasets and Appendix D, Tables D5 and D6 for the two UNSW-NB15 datasets. There and 'A' and 'B' versions of each table with the 'A' version providing the summary of the occurrences of the various Transport Layer protocols and the 'B' version providing the detailed data elements.).

As Table B8 shows, in all datasets except dataSetEAggregator i_dc-9 Dataset, TCP packets comprise over 90% of the total packet count. With the dataSetEAggregator i_dc-9 dataset, however, TCP only comprises 11.7% of the total, with UDP packets making up the remainder. In addition, all the MILCOM 2016 datasets have many TCP packets that are significantly larger than the standard Ethernet packet, which, including the Ethernet header, is a maximum of 1520 bytes (Wright, 2021). Analyzing the various MILCOM 2016 datasets further shows that the captures were all within a single virtual Lan (VLAN). As such, the captures are not limited to the Ethernet total (Wright, 2021). Since this is only representative of system-to-system traffic within a VLAN, all the MILCOM 2016 datasets were eliminated from further analysis.

Table B8 – Candidate Datasets: Transport Layer Protocols

MACCDC 2011 Dataset				
Protocol	**TCP**	**UDP**	**ICMP**	**Remaining 3**
Count	119,292,009	10,259,235	1,690,967	285,469
Percent	90.70%	7.80%	1.29%	0.22%
Capture	28,715,725,818	1,065,182,605	146,353,049	22,162,568
On Wire	28,715,725,818	1,065,182,605	146,353,049	22,162,568
Data	23,997,688,025	672,584,341	68,090,478	11,237,608
Max Cap	1,518	1,518	1,518	78
Max Wire	1,518	1,518	1,518	78
Max Data	1,480	1,480	1,480	40
Min Cap	64	64	64	60
Min Wire	64	64	64	60
Min Data	20	8	8	8
Avg Cap	240	103	86	78
Avg Wire	240	103	86	78
Avg Data	201	65	40	39
MACCDC 2012 Dataset				
Protocol	**TCP**	**UDP**	**ICMP**	**Remaining 2**
Count	68,357,197	560,934	526,103	305,495
Percent	98.0%	0.8%	0.8%	0.4%
Capture	16,302,050,032	62,878,650	37,817,477	23,801,548
On Wire	16,302,050,032	62,878,650	37,817,477	23,801,548
Data	13,593,753,312	41,212,397	13,117,305	12,168,768
Max Cap	1518	1518	582	78
Max Wire	1518	1518	582	78
Max Data	1480	1480	544	40
Min Cap	64	64	64	64
Min Wire	64	64	64	64
Min Data	20	8	8	8
Avg Cap	238	112	71	78
Avg Wire	238	112	71	78
Avg Data	198	73	24	40
UNSW-NB15 17-2-2015 Dataset				
Protocol	**TCP**	**UDP**	**OSPF**	**Remaining 251**
Count	86,029,251	1,430,389	13,776	6,662
Percent	98.349%	1.635%	0.016%	0.01%
Capture	48,178,578,289	163,379,188	1,772,498	2,825,849
On Wire	48,178,578,289	163,379,188	1,772,498	2,825,849
Data	45,078,624,185	111,862,194	1,276,200	2,577,732
Max Cap	1,516	1,520	1,116	1,076
Max Wire	1,516	1,520	1,116	1,076
Max Data	1,480	1,488	1,080	1,032
Min Cap	56	37	60	44
Min Wire	56	37	60	44
Min Data	20	1	24	8
Avg Cap	560	114	128	424
Avg Wire	560	114	128	424
Avg Data	523	78	92	387

dataSetAv2 i-dc-4 Dataset				
Protocol	**TCP**	**UDP**	**ICMP**	
Count	1,576,399	3,843	43	
Percent	99.754%	0.243%	0.003%	
Capture	1,622,224,533	387,275	7,395	
On Wire	1,622,224,533	387,275	7,395	
Data	1,568,626,967	256,613	5,933	
Max Cap	29,026	278	268	
Max Wire	29,026	278	268	
Max Data	28,992	244	234	
Min Cap	66	73	104	
Min Wire	66	73	104	
Min Data	32	39	70	
Avg Cap	1029	100	171	
Avg Wire	1029	100	171	
Avg Data	995	66	137	
dataSetAv2 i-dc-5 Dataset				
Protocol	**TCP**	**UDP**	**ICMP**	
Count	2,121,217	3,551	31	
Percent	99.831%	0.167%	0.001%	
Capture	2,172,273,923	355,770	5,491	
On Wire	2,172,273,923	355,770	5,491	
Data	2,100,152,545	235,036	4,437	
Max Cap	15,994	339	262	
Max Wire	15,994	339	262	
Max Data	15,960	305	228	
Min Cap	54	72	104	
Min Wire	54	72	104	
Min Data	20	38	70	
Avg Cap	1024	100	177	
Avg Wire	1024	100	177	
Avg Data	990	66	143	
dataSetAv2 i-dc-6 Dataset				
Protocol	**TCP**	**UDP**	**ICMP**	
Count	1,421,880	6,157	41	
Percent	99.566%	0.431%	0.003%	
Capture	1,464,242,693	637,190	7,884	
On Wire	1,464,242,693	637,190	7,884	
Data	1,415,898,773	427,852	6,490	
Max Cap	23,234	526	267	
Max Wire	23,234	526	267	
Max Data	23,200	492	233	
Min Cap	54	70	106	
Min Wire	54	70	106	
Min Data	20	36	72	
Avg Cap	1029	103	192	
Avg Wire	1029	103	192	
Avg Data	995	69	158	

dataSetAv2 i-dc-9 Dataset				
Protocol	**TCP**	**UDP**	**ICMP**	
Count	7,486,124	1,038	28	
Percent	99.986%	0.014%	0.000%	
Capture	3,415,415,624	106,009	5,475	
On Wire	3,415,415,624	106,009	5,475	
Data	3,160,887,408	70,717	4,523	
Max Cap	29,026	278	267	
Max Wire	29,026	278	267	
Max Data	28,992	244	233	
Min Cap	66	73	105	
Min Wire	66	73	105	
Min Data	32	39	71	
Avg Cap	456	102	195	
Avg Wire	456	102	195	
Avg Data	422	68	161	

dataSetEAggregator i_dc-9 Dataset				
Protocol	**UDP**	**TCP**		
Count	13,108,792	1,733,279		
Percent	88.322%	11.678%		
Capture	1,722,339,266	1,813,032,879		
On Wire	1,722,339,266	1,813,032,879		
Data	1,276,640,338	1,754,101,393		
Max Cap	1,442	29,026		
Max Wire	1,442	29,026		
Max Data	1,408	28,992		
Min Cap	73	66		
Min Wire	73	66		
Min Data	39	32		
Avg Cap	131	1046		
Avg Wire	131	1046		
Avg Data	97	1012		

At this point, two of the original three MACCDC datasets (MACCDC 2011 and MACCDC 2012) and one of the UNSW-NB datasets (UNSW NB-15 17-2-2015 remain as candidate test datasets. The similarities between the two remaining MACCDC datasets were not as close as those with the two UNSW-NB15 datasets: 1) IPv4 was still the dominant internet layer protocol with approximately 99.9% of the packets; 2) TCP was the dominant Transport Layer protocol, but the percentage of packets in the files was varied (MACCDC 2011: 90.7%, and MACCDC 2012: 98.0%), with UDP being second (MACCDC 2011: 7.8%, and MACCDC 2012: 0.8%); and 3) the data segments for TCP packets were larger than those for UDP, but not as significantly different as with the

UNSW-NB15 dataset (201 and 65 bytes for MACCDC 2011 and 198 and 73 bytes for MACCDC 2012, versus 523 and 78 bytes for the UNSW-NB15 dataset). Unlike the UNSW-NB15 dataset, there was no extra data in any of the two MACCDC datasets. Although the average TCP data segment was larger for MACCDC 2012 dataset, there was not any significant difference between the two, so the MACCDC 2012 was selected over the MACCDC 2011 simply because it was the most recent of the MACCDC dataset capture files available.

Appendix C

MACCDC Datasets

Table C1 - PCAP File Summaries MACCDC 2010 Dataset

PCAP Filename	Total Packets	Total Bytes	Total Bytes On Wire	Total w/Headers	Actual File Size
00000.pcap	10,000,000	765,336,697	765,336,697	925,336,721	925,336,721
00001.pcap	10,000,000	655,887,930	655,887,930	815,887,954	815,887,954
00002.pcap	10,000,000	687,109,057	687,109,057	847,109,081	847,109,081
00003.pcap	10,000,000	728,005,177	728,005,177	888,005,201	888,005,201
00004.pcap	10,000,000	709,419,736	709,419,736	869,419,760	869,419,760
00005.pcap	10,000,000	730,769,304	730,769,304	890,769,328	890,769,328
00006.pcap	10,000,000	738,762,246	738,762,246	898,762,270	898,762,270
00007.pcap	10,000,000	1,044,848,203	1,044,848,203	1,204,848,227	1,204,848,227
00008.pcap	10,000,000	752,930,630	752,930,630	912,930,654	912,930,654
00009.pcap	10,000,000	865,406,857	865,406,857	1,025,406,881	1,025,406,881
00010.pcap	10,000,000	1,935,415,847	1,935,415,847	2,095,415,871	2,095,415,871
00011.pcap	10,000,000	2,568,026,899	2,568,026,899	2,728,026,923	2,728,026,923
00012.pcap	10,000,000	1,598,817,421	1,598,817,421	1,758,817,445	1,758,817,445
00013.pcap	10,000,000	2,043,430,798	2,043,430,798	2,203,430,822	2,203,430,822
00014.pcap	10,000,000	2,484,368,534	2,484,368,534	2,644,368,558	2,644,368,558
00015.pcap	10,000,000	1,453,766,235	1,453,766,235	1,613,766,259	1,613,766,259
00016.pcap	10,000,000	2,279,976,308	2,279,976,308	2,439,976,332	2,439,976,332
00017.pcap	10,000,000	778,115,891	778,115,891	938,115,915	938,115,915
00018.pcap	10,000,000	1,447,111,235	1,447,111,235	1,607,111,259	1,607,111,259
00019.pcap	10,000,000	842,534,398	842,534,398	1,002,534,422	1,002,534,422
00020.pcap	10,000,000	999,195,268	999,195,268	1,159,195,292	1,159,195,292
00021.pcap	10,000,000	1,158,397,077	1,158,397,077	1,318,397,101	1,318,397,101
00022.pcap	10,000,000	1,977,166,435	1,977,166,435	2,137,166,459	2,137,166,459
00023.pcap	10,000,000	711,054,236	711,054,236	871,054,260	871,054,260
00024.pcap	10,000,000	1,007,951,659	1,007,951,659	1,167,951,683	1,167,951,683
00025.pcap	10,000,000	675,318,231	675,318,231	835,318,255	835,318,255
00026.pcap	4,973,151	874,321,512	874,321,512	953,891,952	953,891,952
Totals:	**264,973,151**	**32,513,443,821**	**32,513,443,821**	**36,753,014,885**	**36,753,014,885**

Table C2 - PCAP File Summaries MACCDC 2011 Dataset

PCAP Filename	Total Packets	Total Bytes	Total Bytes On Wire	Total w/Headers	Actual File Size
00000.pcap	10,000,000	1,175,401,003	1,175,401,003	1,335,401,027	1,335,401,027
00001.pcap	10,000,000	955,394,836	955,394,836	1,115,394,860	1,115,394,860
00002.pcap	10,000,000	1,637,376,514	1,637,376,514	1,797,376,538	1,797,376,538
00003.pcap	10,000,000	1,595,081,053	1,595,081,053	1,755,081,077	1,755,081,077
00004.pcap	10,000,000	968,113,473	968,113,473	1,128,113,497	1,128,113,497
00005.pcap	10,000,000	1,223,870,644	1,223,870,644	1,383,870,668	1,383,870,668
00006.pcap	10,000,000	1,661,870,600	1,661,870,600	1,821,870,624	1,821,870,624
00007.pcap	10,000,000	1,673,019,223	1,673,019,223	1,833,019,247	1,833,019,247
00008.pcap	10,000,000	1,258,929,849	1,258,929,849	1,418,929,873	1,418,929,873
00009.pcap	10,000,000	1,072,158,898	1,072,158,898	1,232,158,922	1,232,158,922
00010.pcap	10,000,000	3,748,281,275	3,748,281,275	3,908,281,299	3,908,281,299
00011.pcap	5,000,000	3,064,806,574	3,064,806,574	3,144,806,598	3,144,806,598
00012.pcap	5,000,000	3,343,512,324	3,343,512,324	3,423,512,348	3,423,512,348
00013.pcap	10,000,000	4,270,552,548	4,270,552,548	4,430,552,572	4,430,552,572
00014.pcap	4,465,786	2,589,145,495	2,589,145,495	2,660,598,095	2,660,598,095
Totals:	**134,465,786**	**30,237,514,309**	**30,237,514,309**	**32,388,967,245**	**32,388,967,245**

Table C3 - PCAP File Summaries MACCDC 2012 Dataset

PCAP Filename	Total Packets	Total Bytes	Total Bytes On Wire	Total w/Headers	Actual File Size
00000.pcap	8,635,943	935,501,635	935,501,635	1,073,676,747	1,073,676,747
00001.pcap	4,198,011	1,006,573,119	1,006,573,119	1,073,741,319	1,073,741,319
00002.pcap	2,776,813	1,029,312,773	1,029,312,773	1,073,741,805	1,073,741,805
00003.pcap	1,791,239	1,045,083,217	1,045,083,217	1,073,743,065	1,073,743,065
00004.pcap	3,730,515	1,014,053,658	1,014,053,658	1,073,741,922	1,073,741,922
00005.pcap	3,410,931	1,019,165,726	1,019,165,726	1,073,740,646	1,073,740,646
00006.pcap	2,246,880	1,037,791,936	1,037,791,936	1,073,742,040	1,073,742,040
00007.pcap	2,139,516	1,039,510,245	1,039,510,245	1,073,742,525	1,073,742,525
00008.pcap	3,582,987	1,016,414,006	1,016,414,006	1,073,741,822	1,073,741,822
00009.pcap	3,679,667	1,014,867,161	1,014,867,161	1,073,741,857	1,073,741,857
00010.pcap	4,598,557	1,000,163,983	1,000,163,983	1,073,740,919	1,073,740,919
00011.pcap	4,926,880	994,911,919	994,911,919	1,073,742,023	1,073,742,023
00012.pcap	5,001,472	993,718,688	993,718,688	1,073,742,264	1,073,742,264
00013.pcap	3,190,917	1,022,686,575	1,022,686,575	1,073,741,271	1,073,741,271
00014.pcap	6,763,234	965,530,052	965,530,052	1,073,741,820	1,073,741,820
00015.pcap	7,366,222	955,882,258	955,882,258	1,073,741,834	1,073,741,834
00016.pcap	3,816,907	612,041,887	612,041,887	673,112,423	673,112,423
Totals:	**70,065,452**	**15,658,125,621**	**15,658,125,621**	**16,779,173,237**	**16,779,173,237**

Table C4 - Internet Layer Protocols MACCDC 2010 Dataset

PCAP Filename	Other	IPv4 Protocol	ARP Protocol	Total Packets
maccdc2010_00000.pcap	18	5,078,269	19,287	5,097,574
maccdc2010_00001.pcap	0	9,983,228	2,436	9,985,664
maccdc2010_00002.pcap	0	9,993,162	638	9,993,800
maccdc2010_00003.pcap	0	9,986,589	433	9,987,022
maccdc2010_00004.pcap	0	9,787,705	1,203	9,788,908
maccdc2010_00005.pcap	6	9,562,559	295	9,562,860
maccdc2010_00006.pcap	86	9,854,347	248	9,854,681
maccdc2010_00007.pcap	0	9,966,792	2,059	9,968,851
maccdc2010_00008.pcap	0	9,994,122	267	9,994,389
maccdc2010_00009.pcap	0	9,991,719	404	9,992,123
maccdc2010_00010.pcap	0	9,991,502	918	9,992,420
maccdc2010_00011.pcap	0	9,993,582	427	9,994,009
maccdc2010_00012.pcap	0	9,989,730	610	9,990,340
maccdc2010_00013.pcap	0	9,989,697	494	9,990,191
maccdc2010_00014.pcap	4	9,987,737	744	9,988,485
maccdc2010_00015.pcap	12	9,980,016	1,508	9,981,536
maccdc2010_00016.pcap	0	9,994,944	309	9,995,253
maccdc2010_00017.pcap	0	9,978,262	1,428	9,979,690
maccdc2010_00018.pcap	6	9,579,287	18,075	9,597,368
maccdc2010_00019.pcap	0	9,933,253	4,855	9,938,108
maccdc2010_00020.pcap	6	9,954,496	3,622	9,958,124
maccdc2010_00021.pcap	0	9,970,365	1,268	9,971,633
maccdc2010_00022.pcap	0	9,915,464	3,920	9,919,384
maccdc2010_00023.pcap	0	9,930,754	1,233	9,931,987
maccdc2010_00024.pcap	50	8,181,352	1,701	8,183,103
maccdc2010_00025.pcap	0	9,993,387	204	9,993,591
maccdc2010_00026.pcap	0	4,594,546	2,078	4,596,624
Totals:	**188**	**256,156,866**	**70,664**	**256,227,718**
	0.00007%	**99.9723%**	**0.0276%**	

Table C5 - Internet Layer Protocols MACCDC 2011 Dataset

PCAP Filename	Other	IPv4 Protocol	ARP Protocol	Total Packets
maccdc2011_00000.pcap	1,794	9,446,615	28,962	9,477,371
maccdc2011_00001.pcap	52	9,477,992	1,270	9,479,314
maccdc2011_00002.pcap	168	9,958,454	6,791	9,965,413
maccdc2011_00003.pcap	112	9,774,633	10,613	9,785,358
maccdc2011_00004.pcap	72	9,981,643	3,119	9,984,834
maccdc2011_00005.pcap	113	9,911,204	3,268	9,914,585
maccdc2011_00006.pcap	175	9,958,690	5,246	9,964,111

PCAP Filename	Other	IPv4 Protocol	ARP Protocol	Total Packets
maccdc2011_00007.pcap	1,718	9,072,099	43,854	9,117,671
maccdc2011_00008.pcap	200	9,887,441	11,204	9,898,845
maccdc2011_00009.pcap	238	9,875,155	14,053	9,889,446
maccdc2011_00010.pcap	68	9,984,574	2,289	9,986,931
maccdc2011_00011.pcap	12	4,997,240	358	4,997,610
maccdc2011_00012.pcap	14	4,997,295	351	4,997,660
maccdc2011_00013.pcap	372	9,843,250	11,558	9,855,180
maccdc2011_00014.pcap	273	4,337,905	8,603	4,346,781
Totals:	**5,381**	**131,504,190**	**151,539**	**131,661,110**
	0.004%	**99.881%**	**0.115%**	

Table C6 - Internet Layer Protocols MACCDC 2012 Dataset

PCAP Filename	Other	IPv4 Protocol	ARP Protocol	Total Packets
maccdc2012_00000.pcap	42,316	8,590,643	2,984	8,635,943
maccdc2012_00001.pcap	24,552	4,172,146	1,313	4,198,011
maccdc2012_00002.pcap	16,989	2,758,749	1,075	2,776,813
maccdc2012_00003.pcap	2,854	1,788,228	157	1,791,239
maccdc2012_00004.pcap	27,120	3,701,629	1,766	3,730,515
maccdc2012_00005.pcap	33,336	3,375,939	1,656	3,410,931
maccdc2012_00006.pcap	19,107	2,226,076	1,697	2,246,880
maccdc2012_00007.pcap	15,372	2,122,977	1,167	2,139,516
maccdc2012_00008.pcap	37,551	3,541,248	4,188	3,582,987
maccdc2012_00009.pcap	27,633	3,648,338	3,696	3,679,667
maccdc2012_00010.pcap	17,208	4,578,116	3,233	4,598,557
maccdc2012_00011.pcap	88,142	4,827,213	11,525	4,926,880
maccdc2012_00012.pcap	314,673	4,671,903	14,896	5,001,472
maccdc2012_00013.pcap	65,914	3,116,293	8,710	3,190,917
maccdc2012_00014.pcap	145,136	6,598,950	19,148	6,763,234
maccdc2012_00015.pcap	365,184	6,985,065	15,973	7,366,222
maccdc2012_00016.pcap	762,581	3,046,216	8,110	3,816,907
Totals:	**2,005,668**	**69,749,729**	**101,294**	**71,856,691**
	2.791%	**97.068%**	**0.141%**	

Table C7A - Transport Layer Protocols - MACCDC 2011 Dataset

Protocol Number	Acronym	Protocol	Quantity	Percent of Total
6	TCP	Transmission Control Protocol	119,292,009	90.713%
17	UDP	User Datagram Protocol	10,237,067	7.785%
1	ICMP	Internet Control Message Protocol	1,690,967	1.286%
88	Kerberos	Kerberos Authentication System	277,377	0.211%
2	IGMP	Internet Group Management Protocol	5,286	0.004%
132	SCTP	Stream Control Transmission Protocol	1,484	0.001%
--	--	Remaining Transport Layer Protocols	0	0.00%
		Total:	**131,504,190**	

Table C7B - Transport Layer Protocols - MACCDC 2011 Dataset

Protocol:	1	2	6	17	88	132	Totals/Ranges
Packet Count	1,690,967	5,286	119,292,009	10,237,067	277,377	1,484	**131,504,190**
Total Data	68,090,478	84,120	23,997,688,025	666,018,145	11,095,080	47,488	**24,743,023,336**
Data Segment Min	8	8	20	8	40	32	**8**
Data Segment Max	1,480	24	1,480	1,480	40	32	**1,480**
Data Segment Avg	40	15	201	65	40	32	**66**
Capture Min	64	64	64	64	78	70	**64**
Capture Max	1,518	66	1,518	1,518	78	70	**1,518**
Capture Avg	86	64	240	103	78	70	**107**
On Wire Min	64	64	64	64	78	70	**64**
On Wire Max	1,518	66	1,518	1,518	78	70	**1,518**
On Wire Avg	86	64	240	103	78	70	**107**

Table C8A - Transport Layer Protocols - MACCDC 2012 Dataset

Protocol Number	Acronym	Protocol	Quantity	Percent of Total
6	TCP	Transmission Control Protocol	68,357,197	98.004%
17	UDP	User Datagram Protocol	560,934	0.804%
1	ICMP	Internet Control Message Protocol	526,103	0.754%
88	Kerberos	Kerberos Authentication System	303,549	0.435%
2	IGMP	Internet Group Management Protocol	1,946	0.003%
--	--	Remaining Transport Layer Protocols	68,357,197	98.004%
		Total:	**69,749,729**	

Table C8B - Transport Layer Protocols - MACCDC 2012 Dataset

Protocol:	**1**	**2**	**6**	**17**	**88**	**Totals/Ranges**
Packet Count	526,103	1,946	68,357,197	560,934	303,549	**69,749,729**
Total Data	13,117,305	26,808	13,593,753,312	41,212,397	12,141,960	**13,660,251,782**
Data Min	8	8	20	8	40	**8**
Data Max	544	32	1480	1480	40	**1480**
Data Avg	24	13	198	73	40	**196**
Capture Min	64	64	64	64	78	**64**
Capture Max	582	74	1518	1518	78	**1518**
Capture Avg	71	64	238	112	78	**236**
On Wire Min	64	64	64	64	78	**64**
On Wire Max	582	74	1518	1518	78	**1518**
On Wire Avg	71	64	238	112	78	**236**

Appendix D

UNSW-NB15 Datasets

Table D1 - PCAP File Summaries UNSW-NB15 17-2-2015 Dataset

PCAP Filename	Total Packets	Total Bytes	Total Bytes On Wire	Total w/Headers	Actual File Size
1.pcap	3,526,992	1,943,568,152	1,943,568,152	2,000,000,048	2,000,000,048
2.pcap	3,436,833	1,945,012,012	1,945,012,012	2,000,001,364	2,000,001,364
3.pcap	3,555,838	1,943,107,743	1,943,107,743	2,000,001,175	2,000,001,175
4.pcap	941,588	535,221,182	535,221,182	550,286,614	550,289,571
5.pcap	3,516,448	1,943,737,646	1,943,737,646	2,000,000,838	2,000,000,838
6.pcap	3,626,252	1,941,980,465	1,941,980,465	2,000,000,521	2,000,000,521
7.pcap	3,517,971	1,943,713,647	1,943,713,647	2,000,001,207	2,000,001,207
8.pcap	3,424,905	1,945,201,512	1,945,201,512	2,000,000,016	2,000,000,016
9.pcap	3,618,135	1,942,110,458	1,942,110,458	2,000,000,642	2,000,000,642
10.pcap	3,448,936	1,944,817,053	1,944,817,053	2,000,000,053	2,000,000,053
11.pcap	3,530,922	1,943,505,448	1,943,505,448	2,000,000,224	2,000,000,224
12.pcap	3,532,917	1,943,474,355	1,943,474,355	2,000,001,051	2,000,001,051
13.pcap	3,584,129	1,942,655,174	1,942,655,174	2,000,001,262	2,000,001,262
14.pcap	3,463,943	1,944,576,999	1,944,576,999	2,000,000,111	2,000,000,111
15.pcap	3,514,238	1,943,773,569	1,943,773,569	2,000,001,401	2,000,001,401
16.pcap	3,601,530	1,942,376,198	1,942,376,198	2,000,000,702	2,000,000,702
17.pcap	3,388,055	1,945,792,364	1,945,792,364	2,000,001,268	2,000,001,268
18.pcap	2,037,672	1,259,301,384	1,259,301,384	1,291,904,160	1,291,906,262
19.pcap	2,201,840	1,273,574,851	1,273,574,851	1,308,804,315	1,308,806,331
20.pcap	3,503,117	1,943,950,119	1,943,950,119	2,000,000,015	2,000,000,015
21.pcap	3,548,974	1,943,217,843	1,943,217,843	2,000,001,451	2,000,001,451
22.pcap	3,790,584	1,939,351,032	1,939,351,032	2,000,000,400	2,000,000,400
23.pcap	3,396,023	1,945,663,824	1,945,663,824	2,000,000,216	2,000,000,216
24.pcap	3,570,419	1,942,873,819	1,942,873,819	2,000,000,547	2,000,000,547
25.pcap	3,569,406	1,942,889,641	1,942,889,641	2,000,000,161	2,000,000,161
26.pcap	3,576,768	1,942,772,375	1,942,772,375	2,000,000,687	2,000,000,687
27.pcap	1,067,724	578,989,789	578,989,789	596,073,397	596,074,496
Totals:	**87,492,159**	**48,347,208,654**	**48,347,208,654**	**49,747,083,846**	**49,747,092,020**

Cells highlighted in yellow indicate extra data on file besides packet and PCAP headers.

Table D2 - PCAP File Summaries UNSW-NB15 22-1-2015 Dataset

PCAP Filename	Total Packets	Total Bytes	Total Bytes On Wire	Total w/Headers	Actual File Size
01.pcap	1,800,680	971,189,943	971,189,943	1,000,000,847	1,000,000,847
02.pcap	1,614,980	974,160,455	974,160,455	1,000,000,159	1,000,000,159
03.pcap	1,752,554	971,959,806	971,959,806	1,000,000,694	1,000,000,694
04.pcap	1,736,122	972,222,301	972,222,301	1,000,000,277	1,000,000,277
05.pcap	1,750,127	971,998,776	971,998,776	1,000,000,832	1,000,000,832
06.pcap	1,727,227	972,364,573	972,364,573	1,000,000,229	1,028,295,840
07.pcap	1,825,199	970,797,694	970,797,694	1,000,000,902	1,029,959,340
08.pcap	1,719,835	972,483,651	972,483,651	1,000,001,035	1,000,001,035
09.pcap	1,728,972	972,337,838	972,337,838	1,000,001,414	1,000,001,414
10.pcap	1,879,903	969,922,720	969,922,720	1,000,001,192	1,000,001,192
11.pcap	1,744,141	972,093,795	972,093,795	1,000,000,075	1,000,000,075
12.pcap	1,849,673	970,406,625	970,406,625	1,000,001,417	1,000,001,417
13.pcap	1,741,188	972,141,034	972,141,034	1,000,000,066	1,000,000,066
14.pcap	1,771,257	971,660,451	971,660,451	1,000,000,587	1,000,000,587
15.pcap	1,861,462	970,217,548	970,217,548	1,000,000,964	1,000,000,964
16.pcap	1,876,880	969,969,922	969,969,922	1,000,000,026	1,000,000,026
17.pcap	1,840,858	970,546,450	970,546,450	1,000,000,202	1,000,000,202
18.pcap	1,781,738	971,492,206	971,492,206	1,000,000,038	1,000,000,038
19.pcap	1,735,386	972,233,823	972,233,823	1,000,000,023	1,000,000,023
20.pcap	1,841,677	970,533,351	970,533,351	1,000,000,207	1,000,000,207
21.pcap	1,754,452	971,928,761	971,928,761	1,000,000,017	1,000,000,017
22.pcap	1,771,055	971,663,236	971,663,236	1,000,000,140	1,000,000,140
23.pcap	1,776,553	971,575,264	971,575,264	1,000,000,136	1,014,212,560
24.pcap	1,818,474	970,904,809	970,904,809	1,000,000,417	1,000,000,417
25.pcap	1,819,745	970,884,524	970,884,524	1,000,000,468	1,014,558,428
26.pcap	1,869,147	970,095,118	970,095,118	1,000,001,494	1,000,001,494
27.pcap	1,751,466	971,976,580	971,976,580	1,000,000,060	1,000,000,060
28.pcap	1,777,858	971,554,290	971,554,290	1,000,000,042	1,000,000,042
29.pcap	1,802,883	971,154,332	971,154,332	1,000,000,484	1,000,000,484
30.pcap	1,827,112	970,766,681	970,766,681	1,000,000,497	1,000,000,497
31.pcap	1,942,890	968,913,779	968,913,779	1,000,000,043	1,000,000,043
32.pcap	1,754,452	971,928,761	971,928,761	1,000,000,017	1,000,000,017
33.pcap	1,722,038	972,448,325	972,448,325	1,000,000,957	1,000,000,957
34.pcap	1,793,462	971,305,536	971,305,536	1,000,000,952	1,014,348,648
35.pcap	1,721,390	972,458,786	972,458,786	1,000,001,050	1,000,001,050
36.pcap	1,850,989	970,384,248	970,384,248	1,000,000,096	1,000,000,096
37.pcap	1,853,340	970,347,058	970,347,058	1,000,000,522	1,000,000,522

PCAP Filename	Total Packets	Total Bytes	Total Bytes On Wire	Total w/Headers	Actual File Size
38.pcap	1,870,202	970,077,812	970,077,812	1,000,001,068	1,000,001,068
39.pcap	1,780,903	971,506,560	971,506,560	1,000,001,032	1,000,001,032
40.pcap	1,862,524	970,199,647	970,199,647	1,000,000,055	1,000,000,055
41.pcap	1,975,589	968,390,973	968,390,973	1,000,000,421	1,000,000,421
42.pcap	1,772,799	971,635,209	971,635,209	1,000,000,017	1,000,000,017
43.pcap	1,707,491	972,681,332	972,681,332	1,000,001,212	1,027,982,068
44.pcap	1,728,024	972,351,947	972,351,947	1,000,000,355	1,028,317,452
45.pcap	1,808,596	971,062,774	971,062,774	1,000,000,334	1,029,676,040
46.pcap	1,770,509	971,672,844	971,672,844	1,000,001,012	1,029,000,080
47.pcap	1,811,817	971,011,428	971,011,428	1,000,000,524	1,029,687,268
48.pcap	1,860,601	970,231,234	970,231,234	1,000,000,874	1,030,509,848
49.pcap	1,857,934	970,273,129	970,273,129	1,000,000,097	1,030,485,472
50.pcap	1,860,561	970,231,074	970,231,074	1,000,000,074	1,030,494,104
51.pcap	1,775,797	971,587,417	971,587,417	1,000,000,193	1,029,098,860
52.pcap	1,754,452	971,928,761	971,928,761	1,000,000,017	1,028,743,892
53.pcap	1,186,378	611,006,270	611,006,270	629,988,342	649,457,304
Totals:	**94,571,342**	**51,116,871,461**	**51,116,871,461**	**52,630,014,205**	**53,044,845,688**

Cells highlighted in yellow indicate extra data on file besides packet and PCAP headers.

Table D3 - Internet Layer Protocols UNSW-NB15 17-2-2015 Dataset

PCAP Filename	IPv4 Protocol	ARP Protocol	Total Packets
1.pcap	3,526,347	645	3,526,992
2.pcap	3,436,388	445	3,436,833
3.pcap	3,555,361	477	3,555,838
4.pcap	941,468	120	941,588
5.pcap	3,515,953	495	3,516,448
6.pcap	3,625,790	462	3,626,252
7.pcap	3,517,492	479	3,517,971
8.pcap	3,424,398	507	3,424,905
9.pcap	3,617,604	531	3,618,135
10.pcap	3,448,465	471	3,448,936
11.pcap	3,530,395	527	3,530,922
12.pcap	3,532,414	503	3,532,917
13.pcap	3,583,646	483	3,584,129
14.pcap	3,463,486	457	3,463,943
15.pcap	3,513,775	463	3,514,238
16.pcap	3,601,058	472	3,601,530
17.pcap	3,387,650	405	3,388,055
18.pcap	2,037,415	257	2,037,672
19.pcap	2,201,551	289	2,201,840

PCAP Filename	IPv4 Protocol	ARP Protocol	Total Packets
20.pcap	3,502,638	479	3,503,117
21.pcap	3,548,471	503	3,548,974
22.pcap	3,790,031	553	3,790,584
23.pcap	3,395,604	419	3,396,023
24.pcap	3,569,881	538	3,570,419
25.pcap	3,568,915	491	3,569,406
26.pcap	3,576,303	465	3,576,768
27.pcap	1,067,579	145	1,067,724
Totals:	**87,480,078**	**12,081**	**87,492,159**
	99.99%	**0.01%**	

Table D4 - Internet Layer Protocols UNSW-NB15 22-1-2015 Dataset

PCAP Filename	IPv4 Protocol	ARP Protocol	Total Packets
01.pcap	1,800,426	254	3,601,360
02.pcap	1,614,792	188	3,229,960
03.pcap	1,752,326	228	3,505,108
04.pcap	1,735,914	208	3,472,244
05.pcap	1,749,881	246	3,500,254
06.pcap	1,727,015	212	3,454,454
07.pcap	1,824,951	248	3,650,398
08.pcap	1,719,599	236	3,439,670
09.pcap	1,728,760	212	3,457,944
10.pcap	1,879,651	252	3,759,806
11.pcap	1,743,919	222	3,488,282
12.pcap	1,849,428	245	3,699,346
13.pcap	1,740,978	210	3,482,376
14.pcap	1,771,017	240	3,542,514
15.pcap	1,861,180	282	3,722,924
16.pcap	1,876,590	290	3,753,760
17.pcap	1,840,608	250	3,681,716
18.pcap	1,781,500	238	3,563,476
19.pcap	1,735,173	213	3,470,772
20.pcap	1,841,391	286	3,683,354
21.pcap	1,754,230	222	3,508,904
22.pcap	1,770,825	230	3,542,110
23.pcap	1,776,321	232	3,553,106
24.pcap	1,818,208	266	3,636,948
25.pcap	1,819,481	264	3,639,490
26.pcap	1,868,881	266	3,738,294
27.pcap	1,751,222	244	3,502,932
28.pcap	1,777,646	212	3,555,716

PCAP Filename	IPv4 Protocol	ARP Protocol	Total Packets
29.pcap	1,802,637	246	3,605,766
30.pcap	1,826,884	228	3,654,224
31.pcap	1,942,628	262	3,885,780
32.pcap	1,754,230	222	3,508,904
33.pcap	1,721,792	246	3,444,076
34.pcap	1,793,210	252	3,586,924
35.pcap	1,721,162	228	3,442,780
36.pcap	1,850,721	268	3,701,978
37.pcap	1,853,062	278	3,706,680
38.pcap	1,869,954	248	3,740,404
39.pcap	1,780,647	256	3,561,806
40.pcap	1,862,256	268	3,725,048
41.pcap	1,975,337	252	3,951,178
42.pcap	1,772,567	232	3,545,598
43.pcap	1,707,291	200	3,414,982
44.pcap	1,727,776	248	3,456,048
45.pcap	1,808,360	236	3,617,192
46.pcap	1,770,279	230	3,541,018
47.pcap	1,811,578	239	3,623,634
48.pcap	1,860,333	268	3,721,202
49.pcap	1,857,660	274	3,715,868
50.pcap	1,860,339	222	3,721,122
51.pcap	1,775,589	208	3,551,594
52.pcap	1,754,230	222	3,508,904
53.pcap	1,186,210	168	2,372,756
Totals:	**94,558,645**	**12,697**	**94,571,342**
	99.99%	**0.01%**	

Table D5A - Transport Layer Protocols - UNSW-NB15 17-2-2015 Dataset

Protocol Number	Acronym	Protocol	Quantity	Percent of Total
6	TCP	Transmission Control Protocol	86,029,251	98.34%
17	UDP	User Datagram Protocol	1,430,389	1.64%
89	OSPF	Open Shortest Path First	13,776	0.02%
132	SCTP	Stream Control Transmission Protocol	1,856	0.0021%
1	ICMP	Internet Control Message Protocol	1,594	0.0018%
--	--	Remaining Transport Layer Protocols	3,212	0.0035%
		Total:	**87,480,078**	

Table D5B - Transport Layer Protocols - UNSW-NB15 17-2-2015 Dataset

Protocol:	1	6	17	89	132	Totals/Ranges
Packet Count	1,594	86,029,251	1,430,389	13,776	1,856	**87,477,012**
Total Data	824,684	45,078,624,185	111,862,194	1,276,200	1,497,008	**45,194,099,111**
Data Segment Min	8	20	1	24	12	**1**
Data Segment Max	992	1,480	1,488	1,080	1,020	**1,488**
Data Segment Avg	517	523	78	92	806	**403**
Capture Min	44	56	37	60	48	**37**
Capture Max	1,028	1,516	1,520	1,116	1,056	**1,520**
Capture Avg	557	560	114	128	842	**440**
On Wire Min	44	56	37	60	48	**37**
On Wire Max	1,028	1,516	1,520	1,116	1,056	**1,520**
On Wire Avg	557	560	114	128	842	**440**

Table D6A - Transport Layer Protocols - UNSW-NB15 22-1-2015 Dataset

Protocol Number	Acronym	Protocol	Quantity	Percent of Total
6	TCP	Transmission Control Protocol	93,049,417	98.40%
17	UDP	User Datagram Protocol	1,497,817	1.58%
89	OSPF	Open Shortest Path First	10,137	0.01072%
1	ICMP	Internet Control Message Protocol	433	0.00046%
132	SCTP	Stream Control Transmission Protocol	308	0.00033%
--	--	Remaining Transport Layer Protocols	533	0.00054%
		Total:	**94,558,645**	

Table D6B - Transport Layer Protocols - UNSW-NB15 22-1-2015 Dataset

Protocol:	1	6	17	89	132	Totals/Ranges
Packet Count	433	93,049,417	1,497,817	10,137	308	**94,558,112**
Total Data	211,991	47,600,892,545	109,598,350	571,876	249,356	**47,711,524,118**
Data Segment Min	8	20	1	24	12	**1**
Data Segment Max	992	1,480	1,488	1,080	1,020	**1,488**
Data Segment Avg	489	511	73	56	809	**388**
Capture Min	44	56	37	60	48	**37**
Capture Max	1,028	1,516	1,520	1,116	1,056	**1,520**
Capture Avg	526	547	109	92	845	**424**
On Wire Min	44	56	37	60	48	**37**
On Wire Max	1,028	1,516	1,520	1,116	1,056	**1,520**
On Wire Avg	526	547	109	92	845	**424**

Appendix E

Application Layer Protocols

In the TCP/IP 4-layer network model, Layer 4, the Application Layer, is a combination of the Application, Presentation, and Session layers of the OSI model (Ghosh, n.d.). Transport Layer communications are between two systems and are usually determined by the lowest port number utilized. Therefore, the port usage is determined through simply identifying the lower value of the destination and source ports. The Application Layer protocol is stored in a 4-byte field, so there is a theoretical maximum of 65535 different protocols possible. However, port numbers from 0 to 1023 are considered the well-known ports and reserved for specific protocols. Ports between 1024 and 49151 can be officially registered by IANA for use by a specific application, such as port 5432 which is utilized to communicate with a PostgreSQL database. Finally, ports between 49152 and 65535 are dynamic or private ports, which cannot be reserved, and are utilized by applications for temporary connections between two systems.

Transmission Control Protocol Ports

Table E1A lists occurrences of the top 10 Transmission Control Protocols ports in the MACCDC 2012 dataset. Tables E1B and E1C provide the maximum, minimum, and average packet data sizes for the top 10 TCP ports (Application Layer protocols). Tables E2A and E2B provide the same information for the UNSW-NB15 dataset. The unsupported ports are identified with "(U)" appended to the port number in these tables.

Table E1A – TCP Application Layer Protocol Occurrences – MACCDC 2012

Protocol Number	Acronym	Protocol	Quantity	Percent of Total
80	HTTP	Hypertext Transfer Protocol	11,929,930	17.92%
5,432	--	PostgreSQL	4,489,028	6.74%
54180	--	Unassigned / Dynamic Port	4,183,337	6.28%
443	HTTPS	Hypertext Transfer Protocol Secure	2,382,625	3.58%
22	SSH	Secure Shell	1,973,645	2.96%
902	--	VMware ESXi	1,370,149	2.06%
40064	--	Unassigned / Dynamic Port	1,002,595	1.51%
33643	--	Unassigned / Dynamic Port	646,300	0.97%
445	DS	Directory Services (Active Directory)	511,748	0.77%
36694	--	Unassigned / Dynamic Port	481,148	0.72%
--	--	Remaining 65066 TCP Ports	37,604,011	56.48%
		Total:	**66,574,516**	

Table E1B - TCP Application Layer Protocol Details – MACCDC 2012 – Top 5

Protocol	80	5432	54180(U)	443	22	Totals/Ranges
Count	11,929,930	4,489,028	4,183,337	2,382,625	1,973,645	66,574,516
Capture	4,463,155,891	2,825,646,318	724,022,556	736,728,352	1,213,749,668	15,257,824,196
On Wire	4,463,155,891	2,825,646,318	724,022,556	736,728,352	1,213,749,668	25,987,414,952
Data	3,659,567,591	2,511,405,978	431,168,618	580,449,100	1,077,652,938	20,182,155,318
Max Cap	1,518	1,518	1,518	1,518	1,518	1,518
Max Wire	1,518	1,518	1,518	1,518	1,518	1,518
Max Data	1,460	1,448	1,448	1,460	1,460	1,460
Min Cap	64	64	64	64	64	64
Min Wire	64	64	64	64	64	64
Min Data	0	0	0	0	0	0
Avg Cap	374	629	173	309	614	229
Avg Wire	374	629	173	309	614	390
Avg Data	306	559	103	243	546	303

Table E1C – TCP Application Layer Protocol Details – MACCDC 2012 – Next 5

Protocol	902	40064(U)	33643(U)	445	36694(U)	Totals/Ranges
Count	1,370,149	1,002,595	646,300	511,748	481,148	86,029,251
Capture	1,188,833,508	183,851,524	116,481,582	114,797,729	90,843,266	48,178,578,289
On Wire	1,188,833,508	183,851,524	116,481,582	114,797,729	90,845,474	74,214,877,440
Data	1,106,650,618	113,669,292	72,159,818	79,304,024	57,189,016	71,662,874,855
Max Cap	1,518	1,518	1,518	1,438	1,518	1,516
Max Wire	1,518	1,518	1,518	1,438	1,518	1,516
Max Data	1,460	1,448	1,448	1,380	1,448	1,480
Min Cap	64	64	64	64	64	56
Min Wire	64	64	64	64	64	56
Min Data	0	0	0	0	0	0
Avg Cap	867	183	180	224	188	560
Avg Wire	867	183	180	224	188	863
Avg Data	807	113	111	154	118	833

Table E2A – TCP Application Layer Protocol Occurrences – UNSW-NB15 Dataset

Protocol Number	Acronym	Protocol	Quantity	Percent of Total
6881	--	BitTorrent	19,882,422	23.11%
80	HTTP	Hypertext Transfer Protocol	14,644,588	17.02%
143	IMAP	Internet Message Access Protocol	5,817,783	6.76%
25	SMTP	Simple Mail Transfer Protocol	4,662,046	5.42%
22	SSH	Secure Shell	4,391,772	5.10%
5190	IM	AOL Instant Messenger protocol.	2,442,950	2.84%
21	FTP	File Transfer Protocol	2,266,652	2.63%
179	BGP	Border Gateway Protocol	379,726	0.44%
110	POP3	Post Office Protocol, version 3	250,504	0.29%
445	DS	Directory Services (Active Directory)	164,208	0.19%
--	--	Remaining 57801 TCP Ports	31,126,600	36.18%
		Total:	**86,029,251**	

Table E2B – TCP App Layer Protocol Details – UNSW-NB15 17-2-2015 – Top 5

Protocol	6881	80	143	25	22	Totals/Ranges
Count	19,882,422	14,644,588	5,817,783	4,662,046	4,391,772	86,029,251
Capture	16,461,292,919	13,322,416,872	867,259,833	2,752,278,421	613,430,328	48,178,578,289
On Wire	16,462,547,737	13,322,416,872	867,259,833	2,752,278,421	613,430,328	74,214,877,440
Data	15,108,714,889	12,334,955,596	474,252,834	2,447,509,404	314,435,116	71,662,874,855
Max Cap	1,516	1,516	1,516	1,516	1,516	1,516
Max Wire	1,516	1,516	1,516	1,516	1,516	1,516
Max Data	1,448	1,460	1,460	1,460	1,460	1,480
Min Cap	68	56	56	56	56	56
Min Wire	68	56	56	56	56	56
Min Data	0	0	0	0	0	0
Avg Cap	827	909	149	590	139	560
Avg Wire	827	909	149	590	139	863
Avg Data	759	842	81	524	71	833

Table E2C – TCP App Layer Protocol Details – UNSW-NB15 Dataset – Next 5

Protocol	5190	21	179	110	445	Totals/Ranges
Count	2,442,950	2,266,652	379,726	250,504	164,208	86,029,251
Capture	355,332,281	184,834,610	118,540,486	239,460,235	80,970,227	48,178,578,289
On Wire	355,593,603	184,834,610	118,540,486	239,460,235	80,970,227	74,214,877,440
Data	189,952,232	31,202,986	96,040,232	225,246,388	71,368,518	71,662,874,855
Max Cap	1,516	1,516	1,516	1,516	1,516	1,516
Max Wire	1,516	1,516	1,516	1,516	1,516	1,516
Max Data	1,460	1,460	1,460	1,460	1,460	1,480
Min Cap	56	56	56	56	56	56
Min Wire	56	56	56	56	56	56
Min Data	0	0	0	0	0	0
Avg Cap	145	81	312	955	493	560
Avg Wire	145	81	312	955	493	863
Avg Data	77	13	252	899	434	833

Table E2A shows that the top 10 ports (protocols) of the UNSW-NB15 dataset utilized standard, assigned ports, which was not the case with the MACCDC dataset. Instead, only six (6) of the top 10 ports by occurrence were assigned, while four (4) were unassigned (Table E1A). There was no information provided by the distributor of the dataset, Netresec AB (MACCDC, 2012) concerning this, but tables E1B and E1C shows that these unassigned ports supported a protocol that required a data segment, although not as prevalent as the top assigned ports – 27% of the assigned port average data. Since the structure of these packets followed the TCP format, these differences were ignored, and all packets were included in the program.

User Datagram Protocol Ports

Table E3A lists occurrences of the top 10 User Datagram Protocol ports in the MACCDC 2012 dataset. Tables E3B and E3C provide the maximum, minimum, and average packet data sizes for the top 10 UDP ports (Application Layer protocols). The unsupported ports are identified with "(U)" appended to the port number in these tables. Tables E4A, E4B, and E4C provide the same information for the UNSW-NB15 dataset.

Table E3A – UDP Application Layer Protocol – MACCDC 2012

Protocol Number	Acronym	Protocol	Quantity	Percent of Total
53	DNS	Domain Name System	340,159	60.94%
137	--	NetBIOS Name Service	84,307	15.10%
21371	--	Unassigned / Dynamic Port	13,380	2.40%
161	SNMP	Simple Network Management Protocol	10,526	1.89%
1900	SSDP	Simple Service Discovery Protocol	9,056	1.62%
67	BOOTP	Bootstrap Protocol	7,633	1.37%
5,060	SIP	Session Initiation Protocol	6,784	1.22%
138	--	NetBIOS Datagram Service	6,684	1.20%
57621	--	Unassigned / Dynamic Port	4,418	0.79%
123	NTP	Network Time Protocol	3,355	0.60%
--	--	Remaining 5504 TCP Ports	71,859	12.87%
		Total:	**558,161**	

Table E3B – UDP Application Layer Protocol Details – MACCDC 2012 – Top 5

Protocol	53	137	21371(U)	161	1,900	Totals/Ranges
Count	340,159	84,307	13,380	10,526	9,056	558,161
Capture	31,956,318	8,279,013	1,016,008	957,392	3,078,614	62,440,697
On Wire	31,956,318	8,279,013	1,016,008	957,392	3,078,614	62,440,697
Data	16,305,709	4,400,891	400,528	473,196	2,695,774	36,415,097
Max Cap	346	311	83	346	367	1,518
Max Wire	346	311	83	346	367	1,518
Max Data	300	265	37	300	325	1,472
Min Cap	64	96	68	80	64	64
Min Wire	64	96	68	80	64	64
Min Data	1	50	22	34	-	0
Avg Cap	93	98	75	90	339	112
Avg Wire	93	98	75	90	339	112
Avg Data	47	52	29	44	297	65

Table E3C – UDP Application Layer Protocol Details – MACCDC 2012 – Next 5

Protocol	67	5060	138	57621(U)	123	Totals/Ranges
Count	7,633	6,784	6,684	4,418	3,355	558,161
Capture	2,605,593	2,999,440	1,571,057	397,620	309,640	62,440,697
On Wire	2,605,593	2,999,440	1,571,057	397,620	309,640	62,440,697
Data	2,252,020	2,672,848	1,262,133	194,392	153,924	36,415,097
Max Cap	387	1,421	292	90	94	1,518
Max Wire	387	1,421	292	90	94	1,518
Max Data	341	1,375	246	44	48	1,472
Min Cap	64	64	64	90	64	64
Min Wire	64	64	64	90	64	64
Min Data	-	-	-	44	8	0
Avg Cap	341	442	235	90	92	112
Avg Wire	341	442	235	90	92	112
Avg Data	295	393	188	44	45	65

Table E4A – UDP Application Layer Protocol Occurrences – UNSW-NB15 Dataset

Protocol Number	Acronym	Protocol	Quantity	Percent of Total
53	DNS	Domain Name System	733,276	51.26%
111	ONC RPC	Open Network Computing Remote Procedure Call	321,222	22.46%
4,569	IAX2	Inter-Asterisk eXchange	18,992	1.33%
520	RIP	Routing Information Protocol	15,432	1.08%
137	--	NetBIOS Name Service	10,800	0.76%
514	--	Syslog	5,718	0.40%
5,060	SIP	Session Initiation Protocol	4,826	0.34%
4,433	--	Unassigned / Dynamic Port	1,820	0.13%
69	TFTP	Trivial File Transfer Protocol	854	0.06%
1,608	--	Unassigned / Dynamic Port	330	0.02%
--	--	Remaining 57801 TCP Ports	317,107	22.17%
		Total:	**1,430,377**	

Table E4B – UDP App Layer Protocol Details – UNSW-NB15 Dataset – Top 5

Protocol	53	111	4,569	520	137	Totals/Ranges
Count	733,276	321,222	18,992	15,432	10,800	1,430,377
Capture	70,134,526	40,263,856	1,532,028	3,835,884	1,490,458	163,368,532
On Wire	70,134,526	40,263,856	1,532,028	3,835,884	1,490,458	163,368,532
Data	37,870,822	26,130,088	696,104	3,136,032	1,015,258	100,416,216
Max Cap	850	679	164	1,368	1,132	1,520
Max Wire	850	679	164	1,368	1,132	1,520
Max Data	806	635	120	1,324	1,088	1,480
Min Cap	40	72	56	45	63	37
Min Wire	40	72	56	45	63	37
Min Data	23	28	12	1	19	1
Avg Cap	95	125	80	248	138	114
Avg Wire	95	125	80	248	138	114
Avg Data	51	81	36	203	94	70

Table E4C – UDP App Layer Protocol Details – UNSW-NB15 Dataset – Nest 5

Protocol	514	5,060	4,433(U)	69	1,608(U)	Totals/Ranges
Count	5,718	4,826	1,820	854	330	1,430,377
Capture	2,941,890	3,282,482	167,196	739,007	23,178	163,368,532
On Wire	2,941,890	3,282,482	167,196	739,007	23,178	163,368,532
Data	2,690,298	3,070,138	87,068	702,446	8,658	100,416,216
Max Cap	1,514	1,499	810	1,520	164	1,520
Max Wire	1,514	1,499	810	1,520	164	1,520
Max Data	1,470	1,455	766	1,480	120	1,480
Min Cap	70	68	58	37	68	37
Min Wire	70	68	58	37	68	37
Min Data	26	24	14	4	24	1
Avg Cap	514	680	91	865	70	114
Avg Wire	514	680	91	865	70	114
Avg Data	470	636	47	822	26	70

Unlike the unassigned TCP ports, the unassigned UDP ports had the shortest data segments of the top 10 ports. Regardless, except for the appearance of the Trivial File Transfer Protocol (TFTP) in the UNSW-BN15 dataset, all other assigned/reserved UDP ports are forms of a message format (request and reply packets) and would not lend themselves to data aggregation. Therefore, all UDP packets will be handled as single packets, with the one data segment being processed with the genomic compression technique.

Appendix F

Process Results

Both the MACCDC 2012 and UNSW-NB15 17-2-2015 datasets were processed by the GNCS for various run lengths ranging from 1,000 packets to 25,000,000 packets. For this research the number of packets in a run is hereinafter referred to as "sample size." Analysis of the process output required opening the complete GNCS output file and compiling packet totals and outputting them to a CSV file:

1) Record Number: an artificial record number for sorting purposes.
2) Layer Protocols: three separate values of the Internet, Transport, and Application layer protocols.
3) Data Index: the index into the data section of the output file.
4) Payload: the total of the aggregated data.
5) Packets: the total number of packets in the flow.
6) Capture Length: the length of the raw packet in the PCAP file.
7) With Header: the calculated length of the original packets in the flow.
8) Type: this is an internal reference number identifying the source of the data (completed flow, orphan flow, or edit script).
9) GNCS Size: the size of the data after processing by genomic compression techniques.
10) Ratio: a simple calculation of the savings as shown in the Results Chapter.

The CSV files for each sample are then imported into an Excel spreadsheet and the flowing values are then calculated for each sample:

1) Flow Records: the total number of complete TCP flows.
2) Total Flow Payload: the total of the aggregated data for the flow
3) Total Packets: the total number of packets that make up the flows.
4) PCAP Capture Size: the size of the packet captured, as obtained in the PCAP record header, excluding the header itself.
5) Packet w/Headers: The PCACP Capture Size plus the length of the PCAP record header (16 bytes), plus the length of the PCAP file header (24 bytes).
6) GNCS Size: the total of the final data sizes of each record after processing.

7) Data Savings: this is the savings for the data only, ignoring headers, using the calculation presented in the Results Chapter.
8) GNCS Record Headers: this is the sum of all the possible headers for a flow (52 bytes for the record plus 17bytes for each packet).
9) GNCS Data Header: this is the sum of the 8-byte header required for each data or edit script entry. If the data flow is an exact match for a previous data record, this value would be zero (0).
10) GNCS total: this is the sum of the GNCS size, record header, and data header.
11) Total Savings: this is the calculated space savings for the entire packet.
12) Packets/Flow: this is calculated from the total number of packets divided by the total number of flows.
13) Data/Flow: this is calculated from the total amount of data divided by the total number of flows.
14) Data/Packet: this is calculated from the "Data/Flow" divided by the "Packets/Flow."
15) Compression Ratio: this is the calculated final compression ratio for the data flow.

The results for the MACCDC 2012 and UNSW-NB 17-2-2015 datasets are presented in Tables F1 and F2, respectively.

Table F1 – MACCDC 2012 Results

MACCDC 2012 Dataset Results						
Packets/Run:	1,000	10,000	100,000	1,000,000	10,000,000	25,000,000
Flow Records	118	943	1,301	7,317	103,249	397,645
Total Flow Payload	57,110	541,530	638,132	2,040,003	199,474,195	1,259,098,160
Total Packets	580	5,129	6,630	29,103	495,537	2,227,354
PCAP Capture Size	98,758	908,864	1,112,944	4,136,493	233,869,989	1,405,305,865
Packet w/Headers	108,038	990,928	1,219,024	4,602,141	241,798,581	1,440,943,529
GNCS Data Size	20,001	178,430	265,053	1,645,698	50,690,123	428,704,708
GNCS Data space savings	**65.0%**	**67.1%**	**58.5%**	**19.3%**	**74.6%**	**66.0%**
GNCS Packet Size	36,725	319,667	453,199	2,575,557	64,759,696	488,284,602
GNCS Final space savings	**66.0%**	**67.7%**	**62.8%**	**44.0%**	**73.2%**	**66.1%**
Packets/Flow	4.9	5.4	5.1	4.0	4.8	5.6
Data/Flow	484.0	574.3	490.5	278.8	1932.0	3166.4
Data/Packet	98.5	105.6	96.2	70.1	402.5	565.3
GNCS Compression Ratio	**1.1**	**1.2**	**1.4**	**1.7**	**1.9**	**1.9**

Table F2 – UNSW-NB15 17-2-2015 Results

UNSW-NB15 17-2-2015 Dataset Results						
Packets/Run:	1,000	10,000	100,000	1,000,000	10,000,000	25,000,000
Flow Records	85	382	3,245	21,315	163,073	397,262
Total Flow Payload	112,450	719,913	15,124,305	218,976,259	2,395,911,410	6,033,816,925
Total Packets	317	1,840	28,872	299,384	2,987,288	7,496,751
PCAP Capture Size	131,355	830,711	17,082,715	239,179,210	2,597,907,498	6,540,575,420
Packet w/Headers	136,427	860,151	17,544,667	243,969,354	2,645,704,106	6,660,523,436
GNCS Data Size	112,111	694,458	11,612,820	135,163,974	1,349,231,100	3,418,052,143
GNCS Data space savings	**0.3%**	**3.5%**	**23.2%**	**38.3%**	**43.7%**	**43.4%**
GNCS Packet Size	122,360	747,634	12,282,520	141,441,570	1,409,196,352	1,746,520
GNCS Final space savings	**10.3%**	**13.1%**	**30.0%**	**42.0%**	**46.7%**	**46.4%**
Packets/Flow	3.7	4.8	8.9	14.0	18.3	18.9
Data/Flow	1322.9	1884.6	4660.8	10273.3	14692.3	15188.5
Data/Packet	354.7	391.3	523.8	731.4	802.0	804.9
GNCS Compression Ratio	**1.1**	**1.2**	**1.4**	**1.7**	**1.9**	**1.9**

The results for the two datasets show different trends. The UNSW-NB15 dataset (Table F2) showed an increase in the space savings and compression ratio as the number of packets, and thus the amount of data increases. However, the MACCDC dataset (Table F1) trend was very different in that except for the 1,000,000-packet run, the other runs were relatively like each other, with the 1,000,000-packet run exhibiting a significantly lower space savings and compression ratio.

Appendix G

Acronyms

ACK – Acknowledgement TCP flag
AD – Active Directory
AID – Anomaly-based Intrusion Detection
APT – Advanced Persistent Threat
ARP – Address Resolution Protocol
AU – Audit and Accountability
BBC – Byte-aligned Bitmap Compression
BBST – Balanced Binary Search Tree
BLAST – Basic Local Alignment Search Tool
BLAT – BLAST-like alignment tool
BOOTP – Bootstrap Protocol
CAIDA – Center for Applied Internet Data Analysis
CAC – Computer Access Card
CAMP – Common Affix Merging with Partition
CD – Compact Disc
CIS – Center for Internet Security
CMS – Compact Muon Solenoid
CODIS – Compressing Dirty Snippet
COMPAX – Compressed Adaptive Index
CONCISE – Compound 'n' Composable Integer Set
CSFW – Cybersecurity Framework
CWF = Congestion Window Reduce TCP flag
CUI – Controlled Unclassified Information
DBMS – Database Management System
DCT – discrete cosine transform
DHCP – Dynamic Host Configuration Protocol
DNS – Domain Name Service
DWT – discrete wavelet transform
ECN – Echo TCP flag
EIGRP – Enhanced Interior Gateway Routing Protocol
EWAH – Enhanced Word-Aligned Hybrid
FIN – Finish TCP flag
FISMA – Federal Information Security Management Act
FTP – File Transfer Protocol
GIF – Graphic Interchange Format

GNCS – Genomic Network Compression System
HP – Hewlett Packard
HTTP – Hypertext Transfer Protocol
IAWG – Internet Accounting Working Group
ICMP – Internet Control Message Protocol
IDPS – Intrusion Detection and Prevention System
IDS – Intrusion Detection System
IEC – International Electrotechnical Commission
IEEE – Electrical and Electronics Engineers
IETF – Internet Engineering Task Force
IP – Internet Protocol
IPv4 – Internet Protocol version 4
IPv6 – Internet Protocol version 6
ITL – Information Technology Laboratory
ISO – International Organization for Standardization
JPEG – Joint Photographic Experts Group
LCS – Longest Common String
LZ – Lempel-Ziv
LZ77 – Lempel-Ziv compression algorithm 1977
LZ78 – Lempel-Ziv compression algorithm 1978
LZMA – Lempel–Ziv–Markov
LZW – Lempel-Ziv-Welch compression algorithm
MAC – Media Access Control
MACCDC – Mid-Atlantic Collegiate Cyber Defense Competition
MASC – Maximized Stride with Carrier
MILCOM – Military Communications Conference
MPEG – Moving Pictures Expert Group
MP3 – formally the third audio format of the MPEG-1 standard
MP4 – formally the MPEG-4 Part 15 digital multimedia container format
NIST –National Institute of Standards and Technology
NSA – National Security Agency
OS – Operating System
OSI – Open Systems Interconnection
OWASP – Open Web Application Security Project®
OSPF – Open shortest Path First
PCAP – Packet Capture
PLWAH – Position List Word Aligned Hybrid
PNG – Portable Network Graphics
PWAH – Partitioned Word-Aligned Hybrid
RAR – Roshal Archive file format
RARP – Reverse Address Resolution Protocol
RDBMS – relational database management systems
RST – Reset TCP flag
RLE – run-length encoding
RLH – Run-Length Huffman
SAMI – Sources and Methods Intelligence

ScalaBLAST – BLAST recompiled to utilize multiprocessing
SCTP – Stream Control Transmission Protocol
SECOMPAX – Scope-Extended Compressed Adaptive Index
SEM – Security Event Management
SIEM – Security Information and Event Manager
SIM – Security Information Management
SNMP – Simple Network Management Protocol
SP – Special Publication
SSH – Secure Shell
SQL – Structured Query Language
SYN – Synchronize TCP flag
TAR –a Unix-based utility used to package files together for backup or distribution.
TC – transform coding
TCP – Transmission Control Protocol
TIFF – Tagged Image File Format
TLS – Transport Layer Security
TLS/SSL – Transport Layer Security/Secure Sockets Layer
TLSI – Transport Layer Security Inspection
UCB – Update Conscious Bitmap
UDP – User Datagram Protocol
URG – Urgent TCP flag
URL – Uniform Resource Locator
VAL-WAH – Variable Aligned Length WAH
VPN – Virtual Private Network
WAH – Word-Aligned Hybrid
ZIP – Compression format developed by PKWARE, “ZIP” means “move at high speed”
ZLIB – ZIP Compression Library

References

44 USC 3552, (2017). Title 44 U.S. Code, Sec. 3554, Federal agency responsibilities. 2017 ed. https://www.govinfo.gov/app/details/USCODE-2017-title44/USCODE-2017-title44-chap35-subchapII-sec3554

Aguilera, P. (2006). Comparison of different image compression formats. *Wisconsin College of Engineering, ECE*, 533.

Alaidaros, H., & Mahmuddin, M. (2017). Flow-Based Approach on Bro Intrusion Detection. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 9(2-2), 139-145.

Andersen, R. E. (1981). EDP auditing in the 1980's or the vanishing paper trail. ACM SIGSAC Review, 1(1), 6-15.

Antoshenkov, G. (1995, March). Byte-aligned bitmap compression. In *Proceedings DCC'95 Data Compression Conference* (p. 476). IEEE.

Awati, R. and Scarpati, J. (2022). Deep Packet Inspection (DPI). TechTarget. Retrieved from: https://www.techtarget.com/searchnetworking/definition/deep-packet-inspection-DPI#:~:text=Deep%20packet%20inspection%20(DPI)%20is,only%20packet%20headers%2C%20cannot%20detect.

Berger, B., Daniels, N. M., & Yu, Y. W. (2016). Computational biology in the 21st century: scaling with compressive algorithms. *Communications of the ACM, 59*(8), 72-80.

Berger, B., Waterman, M. S., & Yu, Y. W. (2021). Levenshtein Distance, Sequence Comparison and Biological Database Search. IEEE Trans. Inf. Theory, 67(6), 3287-3294.

Black, D. (2018). Relaxing restrictions on explicit congestion notification (ecn) experimentation (No. rfc8311).

Bowen, T., Poylisher, A., Serban, C., Chadha, R., Chiang, C. Y. J., & Marvel, L. M. (2016, November). Enabling reproducible cyber research-four labeled datasets. In *MILCOM 2016-2016 IEEE Military Communications Conference* (pp. 539-544). IEEE.

Buecker, A., Amado, J., Druker, D., Lorenz, C., Muehlenbrock, F., & Tan, R. (2010). IT Security Compliance Management Design Guide with IBM Tivoli Security Information and Event Manager. IBM Redbooks.

Canahuate, G., Gibas, M., & Ferhatosmanoglu, H. (2007, July). Update conscious bitmap indices. In *19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)* (pp. 15-15). IEEE.

Capon, J. (1959). A probabilistic model for run-length coding of pictures. *IRE Transactions on Information Theory, 5(4)*, 157-163.

CERN. (n.d.). Compact Muon Solenoid (CMS) event dataset. Available from: http://cms.web.cern.ch.

Chambi, S., Lemire, D., Kaser, O., & Godin, R. (2016). Better bitmap performance with roaring bitmaps. *Software: practice and experience, 46(5)*, 709-719.

Chen, Z., Wen, Y., Cao, J., Zheng, W., Chang, J., Wu, Y., ... & Peng, G. (2015). A survey of bitmap index compression algorithms for big data. *Tsinghua Science and Technology, 20(1)*, 100-115.

Chelsea Manning. (n.d.). In nndb.com. Retrieved September 21, 2013, from http://www.nndb.com/people/681/000262892/.

CIS Controls. (n.d.). CIS Controls. Center for Internet Security. Retrieved from: https://www.cisecurity.org/.

Claise, B., Trammell, B., & Aitken, P. (2013). Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information (*No. RFC 7011*).

Colantonio, A., & Di Pietro, R. (2010). Concise: Compressed 'n'composable integer set. *Information Processing Letters, 110(16)*, 644-650.

Colasoft. (n.d.). Colasoft Packet Player. Retrieved from https://www.colasoft.com/packet_player/.

Cuelogic. (2017). The Levenshtein Algorithm. Cuelogic, a Larsen & Toubro Group Company. Retrieved from: https://www.cuelogic.com/blog/the-levenshtein-algorithm.

Deliege, F., & Pedersen, T. B. (2010, March). Position list word aligned hybrid: optimizing space and performance for compressed bitmaps. In *Proceedings of the 13th international conference on Extending Database Technology* (pp. 228-239).

Devopedia. (2019). Levenshtein Distance. Retrieved from: https://devopedia.org/levenshtein-distance.

DNS. (2022). DNS Message Format. GeeksforGeeks. Retrieved from: https://www.geeksforgeeks.org/dns-message-format/

Edpresso. (2022). The Levenshtein distance algorithm. *Educative, Inc*. Retrieved from: https://www.educative.io/edpresso/the-levenshtein-distance-algorithm.

Edward Snowden. (n.d.). National Whistleblower Center. Retrieved September 21, 2013, from https://www.whistleblowers.org/whistleblowers/edward-snowden/.

Elgabry, O. (2017, February 3). Balanced Search Trees. The headache of long paths in BSTs can be treated by self-balancing binary search trees. *Omar Elgabry's Blog*. https://medium.com/omarelgabrys-blog/balanced-search-trees-68a95ba91f50 1/

Embroker. (2021, August 11). *2021 Must-Know Cyber Attack Statistics and Trends*. Embroker Insurance Services, LLC. Retrieved from: https://www.embroker.com/blog/cyber-attack-statistics/

Estan, C., Keys, K., Moore, D., & Varghese, G. (2004). Building a better NetFlow. ACM SIGCOMM Computer Communication Review, 34(4), 245-256.

FBI. (n.d.). What We Investigate. The Cyber Threat. Federal Bureau of Investigation. https://www.fbi.gov/investigate/cyber.

FileInfo. (n.d.). FileInfo.com. The File Format Database. Sharpened Productions. FileInfo.com - The File Format Database.

FISMA. (2002). Federal Information Security Modernization Act (P.L. 113-283), December 2014. https://www.congress.gov/113/plaws/publ283/PLAW-113publ283.pdf.

Fitriya, L. A., Purboyo, T. W., & Prasasti, A. L. (2017). A review of data compression techniques. *International Journal of Applied Engineering Research, 12(19)*, 8956-8963.

Forestiero, A. (2015). A Multi-agent Approach for Intrusion Detection in Distributed Systems. In *Multimedia Communications, Services and Security* (pp. 72-82). Springer International Publishing.

Fusco, F., Stoecklin, M. P., & Vlachos, M. (2010). Net-fli: on-the-fly compression, archiving and indexing of streaming network traffic. *Proceedings of the VLDB Endowment, 3(1-2)*, 1382-1393.

Gartner. (2021, May 17). *Gartner Forecasts Worldwide Security and Risk Management Spending to Exceed $150 Billion in 2021.* https://www.gartner.com/en/newsroom/press-releases/2021-05-17-gartner-forecasts-worldwide-security-and-risk-managem21.

Genomics. (2022, April 30). *Genetic Code*. National Human Genome Research Institute. Retrieved from https://www.genome.gov/genetics-glossary/Genetic-Code.

Ghosh, B. (n.d.). *OSI Network Layer Analysis via Wireshark*. In linuxhint.com. Retrieved From: https://linuxhint.com/osi_network_layer_analsysis_wireshark/.

Gibbons, A. (2012, June 13). *Bonobos Join Chimps as Closest Human Relatives*. American Association for the Advancement of Science. Retrieved from https://www.science.org/content/article/bonobos-join-chimps-closest-human-relatives.

Golovko, V., Vaitsekhovich, L. U., Kochurko, P., & Rubanau, U. S. (2007, August). Dimensionality reduction and attack recognition using neural network approaches. In *IEEE International Joint Conference on Neural Networks, 2007. IJCNN 2007.* (pp. 2734-2739).

Guzun, G., Canahuate, G., Chiu, D., & Sawin, J. (2014, March). A tunable compression framework for bitmap indices. In *2014 IEEE 30th international conference on data engineering* (pp. 484-495). IEEE.

Hamming, R. W. (1950). Error detecting and error correcting codes. *The Bell system technical journal, 29(2)*, 147-160.

Hanson, D. A. (1979, April). Data security. In *Proceedings of the 17th Annual Southeast Regional Conference* (pp. 154-154). ACM.

Harmoush, E. (2019). *OSI Model*. Practical networking. Retrieved from: https://www.practicalnetworking.net/series/packet-traveling/osi-model/.

Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. *Communications of the ACM, 18(6)*, 341-343.

Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., & Pras, A. (2014). Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials, 16(4)*, 2037-2064.

Horne, W. (2015, June). Collecting, Analyzing and Responding to Enterprise Scale DNS Events. In *CODASPY* (p. 73).

Hörz, M. (2020). HxD - Freeware Hex Editor and Disk Editor. Downloaded from: https://mh-nexus.de/en/downloads.php?product=HxD20.

Hosseini, M. (2012). A survey of data compression algorithms and their applications. Network Systems Laboratory, School of Computing Science, Simon Fraser University, BC, Canada.

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE, 40(9)*, 1098-1101.

IANA (2022).IEEE 802 Numbers. *Internet Assigned Numbers Authority*. Retrieved from: https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml.

IANA (2022a). Service Name and Transport Protocol Port Number Registry. *Internet Assigned Numbers Authority*. Retrieved from: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

IBM. (2020). Cost of Insider Threats: Global Report 2020. IBM Security, Retrieved from: https://www.ibm.com/downloads/cas/LQZ4RONE.

ICMP. (2018). ICMP Explained and Packet Format. Cream Magazine by Themebeez. Retrieved from: https://learnduty.com/articles/icmp-explained-and-packet-format/.

IONOS Digital Guide. (2019, October 15). RARP: The Reverse Address Resolution Protocol. https://www.ionos.com/digitalguide/server/know-how/reverse-arp/.

IRS. (n.d.). Security Information and Event Management(SIEM) Systems. United States Internal Revenue Service. Retrieved from: https://www.irs.gov/privacy-disclosure/security-information-and-event-management-siem-systems.

ISO/IEC 27001. (n.d.). ISO/IEC 27001 Information Security Management. International Organization for Standardization. Retrieved from: https://www.iso.org/isoiec-27001-information-security.html.

ISOC (2018, June6). State of IPv6 Deployment 2018, Internet Society. Retrieved from https://www.internetsociety.org/resources/2018/state-of-ipv6-deployment-2018/.

Jokinen, P., Tarhio, J., & Ukkonen, E. (1996). *A comparison of approximate string matching algorithms*. Software: Practice and Experience, 26(12), 1439-1458.

Kalutarage, H. K., Shaikh, S. A., Wickramasinghe, I. P., Zhou, Q., & James, A. E. (2015). Detecting stealthy attacks: Efficient monitoring of suspicious activities on computer networks. *Computers & Electrical Engineering*, *47*, 327-344.

Kavitha, P. (2016). A survey on lossless and lossy data compression methods. *International Journal of Computer Science & Engineering Technology, 7(03),* 110-114.

Kent, K., Chevalier, S., Grance, T., & Dang, H. (2006). Guide to integrating forensic techniques into incident response. *NIST Special Publication 800-86*.

Kerner, S. M. (May 2021.). Internet Protocol (IP). Retrieved from: https://www.techtarget.com/searchunifiedcommunications/definition/Internet-Protocol.

Kim, M. S., Won, Y. J., Lee, H. J., Hong, J. W., & Boutaba, R. (2004). Flow-based characteristic analysis of Internet application traffic. In Workshop Chair.Liao, H. J., Lin, C. H. R., Lin, Y. C., & Tung, K. Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications, 36(1)*, 16-24.

KokiYmgch. (2018). Hirschberg's Algorithm. *KokiYmgch's blog*. Retreived from: https://codeforces.com/blog/entry/57512

Korstanje, J. (2020, January 17). 3 text distances that every data scientist should know. *Towards Data Science.* Retrieved from: https://towardsdatascience.com/3-text-distances-that-every-data-scientist-should-know-7fcdf850e510.

Kyriakopoulos, K. G., & Parish, D. J. (2010). Applying wavelets for the controlled compression of communication network measurements. *IET communications, 4(5)*, 507-520.

Kurose, J. F., & Ross, K. W. (2007). Computer Networking: A Top-Down Approach Edition. Addison Wesley.

Lemire, D., Kaser, O., & Aouiche, K. (2010). Sorting improves word-aligned bitmap indexes. *Data & Knowledge Engineering, 69(1)*, 3-28.

Levenshtein, V. I. (1966, February). Binary codes capable of correcting deletions, insertions, and reversals. *In Soviet Physics Doklady, 10(8)*, pp. 707-710.

Liao, H. J., Lin, C. H. R., Lin, Y. C., & Tung, K. Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications, 36(1)*, 16-24.

LiveAction. (2022). The Ethertype Value Identifier. LiveAction. Retrived from: https://www.liveaction.com/resources/glossary/ethertype-values/.

Loh, P. R., Baym, M., & Berger, B. (2012). Compressive genomics. *Nature Biotechnology, 30*(7), 627-630.

Loh, P. R., Baym, M., & Berger, B. (2012a). Compressive genomics. Supplemental material available at http://www.nature.com/doifinder/10.1038/nbt.2241.

MACCDC. (2012). Capture files from the Mid-Atlantic Collegiate Cyber Defense Competition. National CyberWatch Center. Retrieved from: https://www.netresec.com/?page=MACCDC,

Mahdi, O. A., Mohammed, M. A., & Mohamed, A. J. (2012). Implementing a novel approach and convert audio compression to text coding via hybrid technique. *International Journal of Computer Science Issues (IJCSI), 9(6)*, 53.

Marker, A. (2019, July 19). *Audit Trails: Managing the Who, What, and When of Business Transactions.* https://www.smartsheet.com/audit-trails-and-logs.

Mills, C., Hirsh, D., & Ruth, G. R. (1991). *RFC1272: Internet Accounting: Background*. Network Accounting Group.

Mokalled, H., Catelli, R., Casola, V., Debertol, D., Meda, E., & Zunino, R. (2019, June). The applicability of a siem solution: Requirements and evaluation. In 2019 *IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)* (pp. 132-137). IEEE.

Moustafa, N. and Slay, J. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." *Military Communications and Information Systems Conference (MilCIS)*, 2015. IEEE, 2015.

Mozzherin, D. (2019). damerau-levenshtein. Retrieved from: https://github.com/GlobalNamesArchitecture/damerau-levenshtein

Nanni, D. (2020, November 29). *How to capture and replay network traffic on Linux.* Xmodulo. Retrieved from https://www.xmodulo.com/how-to-capture-and-replay-network-traffic-on-linux.html.

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of molecular biology, 48(3), 443-453.

Networx (n.d.). *EtherType*. Networx Security e.V. Augsburg, Germany. Retrieved from: https://www.networxsecurity.org/members-area/glossary/e/ethertype.html.

NIST CSFW (2018). *Framework for Improving Critical Infrastructure Cybersecurity.* Gaithersburg: National Institute of Standards and Technology. Retrieved from: *https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf.*

NIST ITL Bulletin (1997). Information Technology Laboratory Bulletin 1997-03. Gaithersburg: National Institute of Standards and Technology. Retrieved from: https://csrc.nist.gov/csrc/media/publications/shared/documents/itl-bulletin/itlbul1997-03.txt.

NIST SP 800-37, (2005). *Recommended security controls for federal information systems, Revision 2*. Gaithersburg: National Institute of Standards and Technology. Retrieved from: https://doi.org/10.6028/NIST.SP.800-37.

NIST SP 800-37R2 (2014). *Risk Management Framework for Information Systems and Organizations, Revision 2*. Gaithersburg: National Institute of Standards and Technology. Retrieved from: https://doi.org/10.6028/NIST.SP.800-37r2.

NIST SP 800-39. (2011). *Managing Information Security Risk-Organization. Mission, and Information System View*. Gaithersburg: National Institute of Standards and Technology. Retrieved from: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-39.pdf

NIST SP 800-53. (2005). Guide for the security certification and accreditation of federal information systems. Gaithersburg: National Institute of Standards and Technology. Retrieved from: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53.pdf.

NIST SP 800-53r5. (2020). *Security and Privacy Controls for Information Systems and Organizations, Revision 5*. Gaithersburg: National Institute of Standards and Technology. Retrieved from: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf.

NIST SP 800-92. (2006). Guide to Computer Security Log Management. Gaithersburg: National Institute of Standards and Technology. Retrieved from: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf.

NIST SP 800-94. (2007). Guide to intrusion detection and prevention systems (IDPS). Gaithersburg: National Institute of Standards and Technology. Retrieved from: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-94.pdf.

NIST SP 800-171r2, (2017). *Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations, Revision 2*. Gaithersburg: National Institute of Standards and Technology. Retrieved from: https://doi.org/10.6028/NIST.SP.800-171r2.

Nobel, R. (2018). The Ethertype value, part 1. Stockholm, Sweden: Rickard Nobel, AB. Retrieved from: https://rickardnobel.se/the-ethertype-value-part-1/#:~:text=One%20example%20of%20an%20Ethertype,module%20at%20the%20destination%20host.

Npcap (n.d.). *Packet capture library for Windows.* Nmap.org. Retrieved from https://insecure.org/fyodor/.

NSA. (2019, December 16). *Managing risk from Transport Layer Security Inspection*. National Security Agency Cybersecurity Information. Retrieved October 4, 2021, from https://media.defense.gov/2019/Dec/16/2002225460/-1/-1/0/INFO%20SHEET%20%20MANAGING%20RISK%20FROM%20TRANSPORT%20LAYER%20SECURITY%20INSPECTION.PDF.

O'Dea, S. (2020, June 9). *Business internet traffic volume in the U.S. 2016-2023.* Statista. https://www.statista.com/statistics/995060/business-internet-traffic-in-the-us/

Obama, B. (2011). Executive Order 13587, *National Insider Threat Policy*. Retrieved from: https://www.fas.org/sgp/obama/insider.pdf.

Oehmen, C., Peterson, E., & Dowson, S. (2010, April). An organic model for detecting cyber-events. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research* (p. 66). ACM.

Oehmen, C., Peterson, E., & Teuton, J. (2012, January). Evolutionary drift models for moving target defense. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop* (p. 37). ACM.

OWASP (n.d.). *Intrusion detection*. Intrusion Detection Control | Open Web Application Security Project Foundation. Retrieved October 3, 2021, from https://owasp.org/www-community/controls/Intrusion_Detection.

Pentikousis, K., Badr, H., & Andrade, A. (2010). A comparative study of aggregate TCP retransmission rates. *International Journal of Computers and Applications, 32(4)*, 435-441.

Petryschuck, S. (2019). NetFlow Basics: An Introduction to Monitoring Network Traffic, Auvik.com. Retrieved from: https://www.auvik.com/franklyit/blog/netflow-basics/

Punithavathani, D. S., Sujatha, K., & Jain, J. M. (2015). Surveillance of anomaly and misuse in critical networks to counter insider threats using computational intelligence. *Cluster Computing, 18*(1), 435-451.

Rascu, L. (2015). Hirschberg's algorithm for global string alignment. Retrieved from: https://gist.github.com/lxr/a222b1f063b974d88ce6.

Reghbati, H. K. (1981). Special feature an overview of data compression techniques. *Computer, 14(04)*, 71-75.

Robert Hanssen. (2001, February 20). In www.fbi.gov, *Famous Cases & Criminals.* Retrieved April 24, 2016 from: https://www.fbi.gov/ history/famous-cases/robert-hanssen

Sardari, M., Beirami, A., Zou, J., & Fekri, F. (2013, April). Content-aware network data compression using joint memorization and clustering. In *Proceedings of the 2013 IEEE INFOCOM*, (pp. 255-259). IEEE.

Scarfone, K., & Mell, P. (2007). Guide to intrusion detection and prevention systems (idps). NIST special publication, 800(2007), 94.

Schmidt, A., Kimmig, D., & Beine, M. (2011). A proposal of a new compression scheme of medium-sparse bitmaps. *International Journal on Advances in Software, vol. 4, nos. 3&4*, pp. 401–411, 2011.

Sheldon, R. (2021, August). Enhanced Interior Gateway Routing Protocol (EIGRP). *TechTarget Network*. Retrieved from: https://www.techtarget.com/searchnetworking/definition/EIGRP?Offer=abt_pubpro_AI-Insider.

Shopp, B. (2020, May 9). *Audit trails critical for tracking network activity.* GCN.com. Retrieved from: https://gcn.com/Articles/2020/05/05/audit-trails.aspx?p=1.

Simplilearn. (2023). What Is Kerberos? How Does Kerberos Work: Everything You Need to Know. Simplilearn Solutions. Retrieved from: https://www.simplilearn.com/what-is-kerberos-article#:~:text=Kerberos.

Sindhu, S. S. S., Ramasubramanian, P., & Kannan, A. (2004, November). Intelligent multi-agent based genetic fuzzy ensemble network intrusion detection. In *Neural Information Processing* (pp. 983-988). Springer Berlin Heidelberg.

Singleton, T. W. (1996). EDP auditing in North America: How its development brings insights to contemporary issues.

Smith, A. (2008). Nucleic acids to amino acids: DNA specifies protein. Nature Education, 1(1), 126.

Smith, C. A. (2010). A survey of various data compression techniques. *Int J pf Recent Technol Eng, 2(1)*, 1-20.

Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of molecular biology, 147(1)*, 195-197.

SpeedGuide (2022). speedguide.net. SpeedGuide, LLC. Retrieved from: https://www.speedguide.net/port.php.

Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., & Stiller, B. (2010). An overview of IP flow-based intrusion detection. IEEE Communications Surveys and Tutorials, 12(3), 343-356.

Spring, N. T., & Wetherall, D. (2000). A protocol-independent technique for eliminating redundant network traffic. *ACM SIGCOMM Computer Communication Review, 30*(4), 87-95.

Stabno, M., & Wrembel, R. (2009). RLH: Bitmap compression technique based on run-length and Huffman encoding. *Information Systems, 34(4-5)*, 400-414.

Tcpdump. (2022). Link-Layer Header Types. *The Tcpdump Group*. Retrieved from https://www.tcpdump.org/linktypes.html.

Tcpreplay. (n.d.). Tcpreplay - Pcap editing and replaying utilities. Retrieved from: https://tcpreplay.appneta.com/

Tek-Tools. (2020, February 14). *Intrusion Detection System (IDS) – The Fundamentals*. https://www.tek-tools.com/security/what-is-an-intrusion-detection-system-ids

Teuton, J., Peterson, E., Nordwall, D., Akyol, B., & Oehmen, C. (2013, August). LINEBACkER: Bio-inspired data reduction toward real time network traffic analysis. In *2013 6th International Symposium on Resilient Control Systems (ISRCS)*, pp. 170-174. IEEE.

The TCP/IP Guide. (2005, September 20). *ARP Overview, standards and history*. http://www.tcpipguide.com/free/t_ARPOverviewStandardsandHistory.htm.

UC Berkeley. (n.d.). Security Audit Logging Guideline. The University of California Berkeley, Information Security Office. Retrieved from: https://security.berkeley.edu/security-audit-logging-guideline.

Uthayakumar, J., Vengattaraman, T., & Dhavachelvan, P. (2018). A survey on data compression techniques: From the *perspective of data quality, coding schemes, data type and application*s. Journal of King Saud University-Computer and Information Sciences.

van Schaik, S. J., & de Moor, O. (2011, June). A memory efficient reachability data structure through bit vector compression. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of da*ta (pp. 913-924).

Veritas. (2017, July 28). *Size Matters: A Whole Genome is 6.4B Letters*. Veritas Genetics. Retrieved from https://www.veritasgenetics.com/our-thinking/whole-story/#:~:text=the%20full%20story.-,A%20real%20human%20genome%20is,letters%20(base%20pairs)%20long.

Wang, J., Lin, C., Papakonstantinou, Y., & Swanson, S. (2017, May). An experimental study of bitmap compression vs. inverted list compression. In Proceedings of the 2017 ACM International Conference on Management of Data (pp. 993-1008).

Weinberg, N. (2022). *DHCP defined and how it works*. IDG Communications, LLC. Retrieved from: https://www.networkworld.com/article/3299438/dhcp-defined-and-how-it-works.html.

Welch, T. A. (1984). A technique for high-performance data compression. *Computer, 17(06)*, 8-19.

Wen, Y., Chen, Z., Ma, G., Cao, J., Zheng, W., Peng, G., ... & Huang, W. L. (2014, August). SECOMPAX: A bitmap index compression algorithm. In *2014 23rd International Conference on Computer Communication and Networks (ICCCN)* (pp. 1-7). IEEE.

Wen, Y., Wang, H., Chen, Z., Cao, J., Peng, G., Huang, W. L., ... & Guo, J. (2016, May). MASC: A bitmap index encoding algorithm for fast data retrieval. In *2016 IEEE International Conference on Communications (ICC)* (pp. 1-6). IEEE.

Weinberg, N. (2022). DHCP defined and how it works. *IDG Communications, LLC*. Retrieved from: https://www.networkworld.com/article/3299438/dhcp-defined-and-how-it-works.html.

Wireshark. (2020) Libpcap File Format, Wireshark.org. Downloaded from https://wiki.wireshark.org/Development/LibpcapFileFormat.

Wolfram, S. (2002). *A new kind of science*. Wolfram Media, p. 1069.

Wu, G. (2021). String Similarity Metrics – Edit Distance. *Baeldung.com*. Retrieved from: https://www.baeldung.com/cs/string-similarity-edit-distance.

Wu, Y., Chen, Z., Cao, J., Li, H., Li, C., Wang, Y., ... & Guo, J. (2016). CAMP: A new bitmap index for data retrieval in traffic archival. *IEEE Communications Letters, 20(6)*, 1128-1131.

Wu, K., Otoo, E. J., & Shoshani, A. (2002, July). Compressing bitmap indexes for faster search operations. In *Proceedings 14th international conference on scientific and statistical database management* (pp. 99-108). IEEE.

Wu, K., Otoo, E. J., & Shoshani, A. (2006). Optimizing bitmap indices with efficient compression. *ACM Transactions on Database Systems (TODS), 31(1)*, 1-38.

Wolfram, S. (2002). *A new kind of science*. Wolfram Media, p. 1069.

Zeeh, C. (2003, January). The lempel ziv algorithm. In *URL: http://w3studi. informatik. uni-stuttgart. de/~ zeehca/Seminar/LempelZivReport*. Retrieved from: https://tuxtina.de/files/seminar/LempelZivReport.pdf.

Zheng, Z., & Bockelman, B. (2017, October). Exploring compression techniques for ROOT IO. In *Journal of Physics: Conference Series* (Vol. 898, No. 7, p. 072043). IOP Publishing.

Zheng, W., Liu, Y., Chen, Z., & Cao, J. (2017, May). CODIS: A new compression scheme for bitmap indexes. In *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)* (pp. 103-104). IEEE.

Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on information theory, 23(3)*, 337-343.

Zou, Q., Sun, X., Liu, P., & Singhal, A. (2020). An approach for detection of advanced persistent threat attacks. *Computer*, *53*(12), 92-96.