Noroff University College

Bachelor in
Cyber Security

AN ANALYSIS OF COMPRESSION METHODS FOR SECURITY DATA

TAMÁS SZMANDRA

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR IN **CYBER SECURITY**

SUPERVISOR
Prof. Barry Irwin

Noroff University College, Norway
May, 2024

Mandatory Declaration

Declarations

The individual student is responsible for familiarising themselves with the rules and regulations regarding the use of sources, generated text and academic misconduct. Failure to declare does not release the student from their responsibility.

1.	I hereby declare that the submission answer is my own work, and that I have not used other sources other than as is referenced and cited correctly, or received help other than what is specifically acknowledged.	Yes
2.	**I further declare that this submission:** • Has not been used for another exam in another course at Noroff University College, at another department/university/college at home or abroad. • Does not refer to or make use of the work of others without acknowledgement. • Does not refer to my own previous work unless stated. • Has all the references given in the bibliography. • Is not a copy, duplicate or copy of someone else's work or answer. • Is not generated using AI generation tools.	Yes
3.	I am aware that a breach of any of the above is to be regarded as cheating and may result in cancellation of the exam and exclusion from universities and colleges in Norway, cf. University and College Act §§4-7 and 4-8 and Regulations on examinations §§ 31.	Yes
4.	I am aware that all components of this assignments may be checked for plagiarism and other forms of academic misconduct.	Yes
5.	I hereby acknowledge that I have been taught the appropriate ways to use the work of other researchers. I undertake to paraphrase, cite, and reference according to the acceptable academic practices, in accordance with the rules and guidelines, as taught.	Yes
6.	I am aware that Noroff University College will process all cases where cheating is suspected in accordance with the college's guidelines.	Yes

Publication Agreement

Authorisation for electronic publication of the thesis: Through submission you are accepting that Noroff University College has a perpetual, and royalty free right to retain a copy of work for its own internal use, and has the right to make work publicly available - considering any restrictions to publication.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Barry Irwin, for his unwavering support and helpful guidance throughout the entire thesis process. Composing a thesis demands meticulous attention to detail and a consistent work ethic, often requiring the sacrifice of personal time to delve deeply into the realm of academia. Prof. Irwin's dedication and encouragement have been invaluable.

I also extend my heartfelt appreciation to my beloved partner, Anita, who from the outset encouraged me to embark on this challenging journey and stood by me every step of the way. Her unwavering support and understanding were instrumental in sustaining me through this arduous endeavour, as she tirelessly held our family together while I devoted myself to my studies.

Lastly, I wish to express my profound gratitude to my father, whose unwavering belief in me has been a source of strength and inspiration. His steadfast support and confidence in my abilities have played a pivotal role in shaping me into the person I am today. I am forever grateful for his guidance and encouragement.

Abstract

Security log file data is a vital source that contains detailed historical information about the state of an IT system during a cyber incident. It is imperative to securely preserve them as they are the only sources that can server as the sole means of providing conclusive evidence regarding runtime events. The volume of generated log data is continuously expanding, and without proper management, it may escalate to a critical threshold where it ceases to enhance system capacity or remain viable for retention. Transferring substantial amounts of data, particularly large files, poses constraints on available bandwidth as well. To answer these challenges, this study endeavoured to conduct a statistical analysis on five prevalent compression methods, namely *zip, gzip, bzip2, lz4* and *7z*. Utilising an automated BASH script and leveraging our unique security dataset, *bzip2* is proved to be the most efficient, while *lz4* is the fastest compression algorithm. For a balanced approach between efficiency and speed, *gzip* is recommended. These findings are anticipated to provide valuable insights for security decision-makers, enabling them to manage their security data more judiciously. Consequently, they can conserve time and storage space more effectively.

Keywords: *security log compression, comprehensive compression analysis, compression algorithm: zip, gzip, bzip2, lz4, 7z*

Contents

1 Introduction **1**
1.1 Introduction . . . 1
1.2 Problem Statement . . . 2
1.3 Aims and Objectives . . . 2
1.4 Background and Related Work . . . 3
1.5 Research Methodology . . . 4
1.6 Scope and Limits . . . 5
1.7 Ethical Considerations . . . 5
1.8 Document Structure . . . 5

2 Literature Review **7**
2.1 Introduction . . . 7
2.2 The Fundamental Literature of Compression . . . 7
2.2.1 Essential Books about Compression Methodologies . . . 7
2.2.2 Summarised Journal Article . . . 8
2.3 The Importance of Data Compression . . . 9
2.3.1 Current Data Consumption Statistics . . . 10
2.4 Types of Data Compression . . . 10
2.4.1 Lossy Compression . . . 10
2.4.2 Lossless Compression . . . 10
2.5 The Brief History of Compression . . . 11
2.6 Compression Methods Comparison . . . 13
2.7 Hardware Application Field of Compression . . . 14
2.8 Computer Security Log File Management . . . 15
2.9 Security Log File Retention . . . 16
2.10 Compression techniques for security log management . . . 17
2.11 Summary . . . 17

3 Research Approach **19**
3.1 Introduction . . . 19
3.2 Compression Tool Selection . . . 19
3.2.1 Zip . . . 20
3.2.2 Bzip2 . . . 20
3.2.3 Gzip . . . 20
3.2.4 Lz4 . . . 21
3.2.5 7-Zip . . . 21
3.3 Test Plan . . . 22
3.4 The Test Environment . . . 22
3.4.1 The Data Sets . . . 22
3.5 The Testing Framework . . . 23
3.5.1 BASH Script Execution Flow . . . 24
3.5.2 General Flow of Execution . . . 25
3.5.3 Measuring and Metrics . . . 25
3.5.4 Expected Results . . . 25
3.5.5 Processing and Analysing the Results . . . 26
3.6 Summary . . . 26

4 Data Compression Evaluation **27**
4.1 Introduction . . . 27
4.2 Security Log Corpus . . . 28
4.2.1 Access Log Dataset . . . 30
4.2.2 Apache HTTP Log Dataset . . . 30
4.2.3 Connection Log Dataset . . . 30
4.2.4 Csvhoneypots Dataset . . . 31
4.2.5 Email Dataset . . . 31
4.2.6 Files Log Dataset . . . 32
4.2.7 Honeypots JSON Dataset . . . 32
4.2.8 HTTP Log Dataset . . . 32
4.2.9 Network Traffic Dataset . . . 33
4.2.10 SNORT Log Dataset . . . 33
4.3 Summary . . . 34

5 Discussions **36**
5.1 Introduction . . . 36
5.2 Experiment Overview . . . 37
5.3 Preparation . . . 37
5.4 Units and Measurements . . . 38
5.4.1 Compressed Size . . . 38
5.4.2 Compression Rate . . . 38
5.4.3 Average Compression Ratio . . . 38
5.4.4 Compression Time (s) . . . 39

5.4.5 Average Compression Time (s) . . . 39
5.4.6 Compression Speed (MB/s) . . . 39
5.4.7 Average Compression Speed (MB/s) . . . 40
5.4.8 Decompression Time (s) . . . 40
5.4.9 Average Decompression Time (s) . . . 40
5.4.10 File Entropy . . . 41
5.5 Data Visualisation . . . 41
5.5.1 Statistical data tables . . . 42
5.5.2 Graphs and charts . . . 42
5.5.3 Quadrants . . . 43
5.5.4 Byte Value Histogram . . . 43
5.6 Zip Statistical Analysis . . . 44
5.6.1 Zip Compression Performance Analysis . . . 44
5.6.2 Zip Decompression Performance Analysis . . . 46
5.6.3 Zip Quadrants Analysis . . . 46
5.6.4 Zip Byte Value Histogram Analysis . . . 47
5.6.5 Summary . . . 48
5.7 Bzip2 Statistical Analysis . . . 48
5.7.1 Bzip2 Compression Performance Analysis . . . 48
5.7.2 Bzip2 Decompression Performance Analysis . . . 50
5.7.3 Bzip2 Quadrants Analysis . . . 50
5.7.4 Bzip2 Byte Value Histogram Analysis . . . 51
5.7.5 Summary . . . 52
5.8 Gzip Statistical Analysis . . . 52
5.8.1 Gzip Compression Performance Analysis . . . 52
5.8.2 Gzip Decompression Performance Analysis . . . 54
5.8.3 Gzip Quadrants Analysis . . . 54
5.8.4 Summary . . . 55
5.9 Lz4 Statistical Analysis . . . 55
5.9.1 Lz4 Compression Performance Analysis . . . 55
5.9.2 Lz4 Decompression Performance Analysis . . . 57
5.9.3 Lz4 Quadrants Analysis . . . 57
5.9.4 Summary . . . 58
5.10 7z Statistical Analysis . . . 58
5.10.1 7z Compression Performance Analysis . . . 58
5.10.2 7z Decompression Performance Analysis . . . 60
5.10.3 7z Quadrants Analysis . . . 60
5.10.4 Summary . . . 61
5.11 Overall Performance Evaluation . . . 62
5.11.1 Overall Speed Performance . . . 62
5.11.2 Overall Compression Efficiency . . . 63
5.11.3 Quadrants Evaluation . . . 63

5.11.4 Recommendation 65

6 Conclusion **66**
6.1 Introduction 66
6.2 Summary of Research 66
6.3 Research Objectives 67
6.4 Research Contribution 67
6.5 Future Work 68

References **69**

A Datasheet **75**

List of Figures

2.1 Data compression history (ETHW, 2011) 12

5.1 Zip compression performance 45
5.2 Zip compression efficiency 45
5.3 Zip performance quadrants 47
5.4 Phishingmail.csv before compression 47
5.5 Phishingmail.csv after compression 47
5.6 Bzip2 compression performance 49
5.7 Bzip2 compression efficiency 49
5.8 Bzip2 performance quadrants 51
5.9 Snortlog.pcap before compression 51
5.10 Snortlog.pcap after compression 51
5.11 Gzip compression performance 53
5.12 Gzip compression efficiency 53
5.13 Gzip performance quadrants 54
5.14 Lz4 compression performance 56
5.15 Lz4 compression efficiency 56
5.16 Lz4 performance quadrants 57
5.17 7z compression performance 59
5.18 7z compression efficiency 59
5.19 7z performance quadrants 61
5.20 Overall performance quadrants 64

List of Tables

3.1 List of test algorithms 20
3.2 Compression methods with syntax 21

4.1 Log data structure 28
4.2 Security log corpus 34

5.1 Zip compression metrics 45
5.2 Bzip2 compression metrics 49
5.3 Gzip compression metrics 53
5.4 Lz4 compression metrics 56
5.5 Lz4 compression metrics 59
5.6 Overall performance speed 62
5.7 Overall performance efficiency 63
5.8 Overall performance efficiency 65

CHAPTER 1

Introduction

1.1 Introduction

Data emission and consumption ensure the information flow continuity in the world of information technology. Immeasurable amount of devices communicating through billions of nodes between continents. Enormous amount of data exchange location on daily basis. The start-and endpoints of this multiple way communication are physical devices with limited capabilities. The capability of the network as an intermediary between these nodes is even more restricted. Hence its performance very much depends on the quantity of data being transferred throughout. The data either static and resides on a physical device or in transit between multiple nodes. These big data carries vital information to its owner and every byte of data has a direct cost.

Today's current information flow based on enormous amount of data created, modified and transferred between end users on daily basis (Taylor, 2023). A prerequisite of storing data can be measured in relocatable or free spaces. Technically each device has a production cost that depends on the actual technology being used to produce them. With other words storing and transferring enormous amount of data can be expensive. Especially if one will utilise high performance involving the most modern technology in order to ensure data flow and security best possible way.

From security point of view archiving and preserving data for long term is essential. Security system monitoring is crucial to provide organised approach to minimise and prevent upcoming incidents (Tugnarelli et al., 2018). Correspondingly, any data loss can lead to critical consequences. Therefore in practice security data (log files, reports etc.) is required to be saved and stored on daily basis. Above this, the stored information needs to be moved to archive data storage. If necessary these data - some are stored for ages - should be available and must be rolled back to its original state without any further data

loss. As an example, the work of forensics analysis very much depends on data archiving techniques and guidelines (Brezinski & Killalea, 2002) followed by during and after security incidents.

This study is willing to address solutions for the challenges of security data preservation on-site and under transfer. The following questions might arise are: What are the possible compression techniques to preserve these data in the most efficient manner? How can we optimise our physical storage space by using them? What are the most common compressing techniques and their performance ratio? What are their application fields? What are the best, the fastest and the most reliable compression-decompression techniques for security data archiving?

Data compression is a method based on mathematical algorithm to reduce the size of data, while preserving the essential components and information contained in it (Sayood, 2017). Data compression mechanism can be used in many real-life scenario scaling from text file archiving to preserve video and audio files, through important log files that - from a security perspective - can store vital system information.

In order to address solution to the above mentioned problems, this study offers a comprehensive data compression analysis, based on publicly available data set by utilising the most effective compression methods. To understand these concerns in more depth the next section will briefly present the scientific background for this thesis.

1.2 Problem Statement

Raw data consumes significant storage and poses challenges for both storage and efficient transfer due to space limitations and network bandwidth constraints. Storing extensive amounts of uncompressed data is not only costly but also raises security challenges.

1.3 Aims and Objectives

The primary objective of this research is to provide reasonable and tested data archiving procedures in order to optimise data both on-premise and during transfer. A provided data set is going to be compressed and decompressed with the proposed compression techniques and their performance will be analysed based on speed, latency and redundancy. Finally each compression method will be categorised in accordance to their test results to outline and compare their relative position in terms of speed and efficiency.

The following sub-objectives have been identified in scope in order to achieve the primary objective:

- Identify the difficulties/challenges in regards of storing and transferring large amount of security data both on individual level as well as on organisational basis. Identify security policies in terms of data archiving and outline best practices

- Create security data corpus including security log files, network traffic files, IDS files etc. Create the test environment.
- Write a script to create an automated test environment.
- Identify and select different compression methods (*zip, gzip, bzip2, lz4, 7z*) and conduct research by examining their compression-decompression performance, speed, compression ratio, moreover the average of these metrics. Additional check on their binary level and entropy should be taken.
- Measure, analyse and evaluate the performance of compression algorithms and provide a ranking list.
- Categorise and present the findings and determine the fastest and most accurate compression method for security log file archiving. Considering the findings give recommendation about the application field of each compression tool.

1.4 Background and Related Work

Huffman (1952) has introduces to the world a groundbreaking compression technique that was a revolutionary step in data science in the late '40s and still remains a fundamental concept of compression and decompression techniques. This paper contains the basic technical knowledge is necessary to understand how text compression mechanisms are working. Hazboun and Bassiouni (1982) introduces the reader to pure mathematical concept of the "reversible semantic-independent variable-length character encoding" constructed on Huffman-encoding hierarchy. Interesting to notice that this paper dates back to the '80s and strongly arguing about the need for efficient compression techniques.

In one of the most essential book of data compression Sayood (2017) the author provides a well-detailed explanation about the theory of compression methods, their underlying mathematical substances and their function in real-life data preservation. The book guides us through the essential base of compression algorithm such as Huffman-coding, the different types of arithmetic coding as well image and audio compression and decompression mechanism. The initial knowledge of the compression methods inevitable to understand the basic concepts of algorithm coding in term of compression mechanism. Mohammed and Emary (2007) also published a comparative journal article about various algorithm techniques, where they have emphasised the importance of data compression due to limited storage space and restricted bandwidth. The main area of this work is to compare various compression algorithm by obtaining commonly used files (*.doc, .txt, .jpg etc.*). The result summarises the most efficient algorithm in regards compression ratio and compressed file size. A well-detailed work from Otten et al. (2007) outlines an evaluation about data compression methods in relation to NGN (Next Generation Network). This journal article based on experimental research technique of different types of compression methods.

To get more insight into the basic definition of security data Yang et al. (2020) introduce a comprehensive review of the literature on data security and privacy issues. This journal article focuses on highlighting the challenges and requirements of data security and privacy protection in relation to cloud security. In connection to security log data analysis, different types of compression methods are summarised in the journal article, written by Radley (2015). It also introduces a pseudo-random approach for log data maintenance and file compression performance penalties encountered during the research.

A well-written book of log management (Schmidt et al., 2012) let the reader have more insight into the world of logging laws and mistakes, best practices, cloud logging etc. which defiantly should worth to investigate further. A conference paper from Tugnarelli et al. (2018) explores two types of digital data collection methods for comparative analysis of web server log management.

In relation to considerable policies to evidence collection and data archiving an RFC3227 paper from Brezinski and Killalea (2002) give useful guidelines to security system administrator. Within this short but concise paper they propose the basic procedures must take into consideration during evidence collection and data archiving. Another but more detailed work about the same topic can be obtained from Kent and Souppaya (2006), under the wings of NIST (National Institute of Standards and Technology), that has a complete guide to Computer Security Log Management. The authors clarify the definition of security log and its structure and importance in information security systems. The goal of digital preservation techniques and their application fields has been discussed in the work of (Lee et al., 2002). This research journal article includes surveys, study cases that provide insight the advantages and disadvantages of archiving strategies.

1.5 Research Methodology

This empirical research methodology will demonstrate various compression and decompression methods on large size of security data, that is collected from diverse, publicly available resources. The subjects of this research are security log files, network traces, netflow and IPFIX records, while in some other cases IDS (Intrusion Detection System) or IPS (Intrusion Prevention System) log files. These security files preserved in daily basis and accumulated after a period of time. Their size scales from tens to hundreds of megabytes on daily basis. The chosen compression algorithms will be applied on the very same size of files regardless. During the whole examination procedure the performance of each compression algorithm is going to be recorded and measured. The collected numerical metrics will be further examined. Finally we will assigned positions to each compression tool, sorting them by different criteria. As the last phase, all the collected information are going to be evaluated and visually presented.

1.6 Scope and Limits

This analysis focuses on publicly available compression methods, applied to security log files and outlines the following specific areas that are covered within the scope of examination:

- Evaluation of diverse commonly used compression methods on text-based security log data including and limited to the following types: *zip, gzip, bzip2, lz4, 7z*.
- Examination of metrics in terms of performance that includes compression ratio, compression speed, decompression speed and the average of these.
- Integration of these methods to automation environment.
- Identification of security log file best practices and recommendation for storage space and speed optimisation.

While this study aims to provide comprehensive security log data compression analysis, there are certain aspects that fall outside the scope of this research:

- Detailed examination of proprietary or special log file formats.
- Evaluation of streaming compression methods and lossy compression methods.
- Discussion of encrypted data, video, sound or any graphical object compression.
- Detailed examination of OS specific compression.
- Investigation and analysis of hardware-based resource utilisation.

1.7 Ethical Considerations

The process of collecting, analysing and interpreting security log file data may raise some ethical consideration first and foremost about anonymity and confidentiality. Along the data collection process we aimed to use trustworthy sources that have reliable background and also apply written consent that their data can be freely processed and distributed. Furthermore, the data collected, created an analysed in this work are only processed by the author. The data evaluation process does not in any way harm unintended persons.

1.8 Document Structure

Within this section the reader can have a brief insight of the entire structure of this study, that is for the sake of simplicity is divided and described in the following bullet points.

- **Abstract**
  Defines an overview of the thesis, clearly summarises the essence of the thesis.
- **Introduction**
  Introduces the research topic, defines the scope of study, announce the problem statement and informs the reader about the background of the research. Outlines the structure of the work and provides the necessary ethical considerations.

- **Literature Review and Theoretical Background**
  Represents the essential literature in connection to the assigned subject and outlines the core concepts of the research conducted. Establish the theories behind the research.
- **Research Methodology**
  Initiates the methods were tested, the test framework and the test plan, moreover outlines the expected outcomes.
- **Data Compression Evaluation**
  Introduces the subject of the investigated compression methods and briefly describes their structure and function.
- **Discussions**
  Summaries the goal of the research, outlines the body of the statistical approach is investigated, discuss examined methods more in depth and evaluates the findings and provides recommendations.
- **Conclusion**
  Brings together the evaluated findings, summarises the results and provides insight for future developments.

CHAPTER 2

Literature Review

2.1 Introduction

The following literature review presents the core value of this thesis. Each section aims to outline the subject in more depth and identify the role of compression mechanisms in daily life. The first section (**??**) intends to guide the reader through some of the most essential literature in the field of compression algorithm. After a brief introduction to the importance of data compression (2.3) the different types of compression algorithm (2.4) is discussed. As we approach further, we can have a bit more insight how the history of compression (2.5) is evolved. Some technical aspect are taken place in section 2.6, and 2.7 to build a solid background for the upcoming technical part. Section 2.8 and 2.9 highlights the aspects of security log file management and retention respectively.

2.2 The Fundamental Literature of Compression

Research studies in general are build on others experience and works undertaken in the academic field. The following sections summarise the corresponding literature that are essential part of this paper.

2.2.1 Essential Books about Compression Methodologies

Each research field has their own fundamental academical resources that aims to grant a solid basis for researchers. The literature of compression methodologies are neither an exception of this. David Salomon (2002) provides a rich insight in the history and basics of compression techniques, also provides a comprehensive understanding of lossy and lossless compression based on discrete mathematical concepts. The author breaks down compression methods to its elemental level (source coding) and explain in a clear manner

what is the main goal of applying compression techniques. He also touches image, video and audio compression forms, methods and their application field in daily life.

Another reader-friendly book is from Salomon (2007) explains the foundation and main approaches of data compression. It present the most popular algorithms and essential concepts of run-length encoding, variable-length coding etc. The reader can get a clear overview of the main principle of compression methods through multiple examples and additional web-based resources with auxiliary material.

Pu (2006) starts with a brief history of data compression and explains the differences between lossy and lossless compression, their quality measurement an limitation. The author leads the reader through symbolic coding models, prefix coding and entropy in coding, nevertheless presents variable coding algorithm in depth. This book provides a deep insight to the principle of the most important coding schemes that are still in use nowadays, and contains several laboratory-based examination and written in a clear and concise manner. It provides useful exercises for the readers to comprehensively test their knowledge and compare various algorithm.

Prof. Khalid Sayood has written two books about the science of data compression. In *Introduction to Data Compression*, Sayood (2017) provides a wide and well-detailed mathematical description of lossless compression and its model-and coding schemes. He devotes nearly fifty pages to the Huffman coding algorithm, continues with arithmetic coding and dictionary techniques. It is hard to find a compression coding mechanism that has not been mentioned in this work. His other book (Sayood, 2002) comprises the major concept of information theory behind source coding as well as the characteristics of numerous universal coding schemes. The author assigns an entire chapter to comparative analysis.

2.2.2 Summarised Journal Article

It is very rare to find a journal article that includes a wide range of surveys about state-of-the-art data compression and encompass several aspect about the research subject. However the comparative analysis of Jayasankar et al. (2021) represent a solid review of surveys about different data compression techniques (Holtz, 1993), (Srisooksai et al., 2012), classification reviewed resources based on data quality (Drost & Bourbakis, 2001), coding schemes (Capon, 1959), data type (Abel & Teahan, 2005), (Kalajdzic et al., 2015) and application (Kolo et al., 2012), (Ruxanayasmin et al., 2013). This work is one of the most rigorous journal article can be found about data compression with the intention to summarise all available research papers in one. It forms a collection of data compression literature, discusses potential issues and indicates future directions along the subject. Moreover it discretely touches the role of compression in machine learning as well (Abu Alsheikh et al., 2016). The author highlights that due to the main characteristics of data compression such as compatibility, computational complexity or memory management it is important to choose the adequate representation form to avoid data loss or high overhead. This is especially valid for real-time applications as this field

suffers the most of the disadvantage of extensive resource consumption. Even thought some compression perform well in terms of compression ratio, they are far from handling memory management the most efficient way.

2.3 The Importance of Data Compression

Since 1990's the amount of data that has been used and stored in the area of molecular biology doubled the size in 18 months (Crochemore & Lecroq, 1996). This is an example of text processing where large amount of data is stored in linear files. Pattern matching and text-based data compression comes handy in these situations and indicates the need for efficient data compression algorithm, even in case when the hardware storage capacity continuously increase.

The ever growing mass data consumption forced both users and Internet Service Providers (ISP) to find new approaches for more effective communication, higher speed of data transfer and different approaches to maintain data storage. The latter has been major concern ever since databases exist and e.g. text documents, legal and medical information needs to be archived. In conjunction to this, in the filed of genomics research enormous amount of data is produced that can reach 40 exabytes yearly by the year of 2025 (Greenfield et al., 2019). Storing research related big data especially challenging. Modern big data systems create massive amount of complex data (Statista, 2024).

Data compression is a common requirement in nowadays's application infrastructure and database systems as well. Iyer et al. (1994) examines the cost effect of using I/O subsystems in relation to Database Management System (DBMS) application and software vs. hardware-based compression technologies. The author concludes that applying lossless compression methods on DBMS applications and subsystems save not only disk space, but significantly reduce their operational cost. We can see that data compression are in use within various fields of our daily life and particularly important to handle maintenance issues with large amount of data, reducing costs and improving data availability.

Data compression has a significant role to reduce and optimise available storage space especially in those scenarios where huge amount of data needs to be stored (data centers, cloud storage, archival systems etc.). From the perspective of disaster recovery, compressed data can be easier replicated or backed up. During transmission, compressed data requires less bandwidth. Intern corporate network's bandwidth scales poorly, therefore compressed data can make the transmission smoother and improve network performance. All in all, data compression can support the overall performance of the system, optimise the amount of transferred data, reduce storage and transmission costs, and nevertheless enhance system scalability.

2.3.1 Current Data Consumption Statistics

In our daily life, without data compression utilisation a simple audio file might consume 100 MB of storage space and a 10 minutes video would easily occupy 1 GB of space (Sardar Ali, 2021). Further statistics (Duarte, 2023) shows the tendency of data consumption of last year (2023). The result symbolise an intensive increasing tendency in daily data usage. The most common fields are certainly connected to social media, video streaming, digital photo sharing and IoT device communication (Marr, 2024). Each of the latter mentioned fields utilises some sort of data compression method in order to preserve data structure as well as to secure smoother data transmission during communication.

2.4 Types of Data Compression

Data compression is a fundamental technique for minimising the inner structure of any given data and by doing that optimising the whole data structure in an efficient way to consume less space and be compact enough to enhance transmission speed during transport. Generally, compression methods are divided into two broad categories: lossless and lossy compression. Kavitha (2016) gives a theoretical overview of the advantages and disadvantages of lossless compression in different scenarios. While lossy compression performs better on audio and video files, lossless compression is more suitable for text. However, the principle of lossless text data compression also applies to video and image compression (Rahman & Hamada, 2019) where for instance Huffman coding outperforms other general encoding methods.

2.4.1 Lossy Compression

Lossy compression, due to the nature of compression methods, compresses the data by discarding or with other words losing some portion of the data during data compression transmission (Souley et al., 2014). In this scenario the loss of data is acceptable even though that reconstructing the original message/data can only provide approximate result (Hosseini, 2012). Hussain et al. (2018) explicitly outlines that the main goal of lossy compression is to shrink the contained information to represent the fewest amount of bits and on the same time reproduce the information without any additional loss. Lossy compression methods are most commonly applies on image (*jpg*), audio (*mp3*) and video (*x264*) format as their application fields do not require reversible output of the original data.

2.4.2 Lossless Compression

Sayood (2002) builds his entire book around lossless compression theory and explains the importance of lossless compression, where any loss of the original data can cause unwanted consequences. It is quite rare that during the process of text compression any loss of data can be tolerated. Mohammed and Emary (2007) supports this thoughts as

loosing any text character during the compression or decompression process makes the text misleading. Accordingly, even image or audio file compression should generally be compressed by lossless algorithm to achieve better quality, but there are certain limitation which compression method can be applied on a single file. Nevertheless, there is a trade off in terms of size.

Smith (2010) wrote a survey focused on the mathematical aspects - mainly linear algebra - of data encoding. The core part of his work represents how signals and waves in the nature correlates with e.g. Fourier Transform or Haar wavelets compression. Several comparative studies can be found in relation to both lossless text data comparison. Although the previously mentioned approach seems to be a bit abstract, a state-of-the-art surveys from Singh and Singh (2012) introduces the major compression through basic examples. Run-length coding, Huffman coding and the Lempel-Ziv algorithm are visualised and explained in a clear manner. It is concluded that some algorithms like Burrows-Wheeler Transform (BWT) or Move-to-front Transform (MFT) do not compress, only serve as an input to to be compressed at a later phase.

2.5 The Brief History of Compression

Salomon (2002) wrote about an interesting preface about the Braille blind coding, dates back to 1829, where each character of the alphabet were replaced with a shorter, more compressed form (raised or flat dots on a paper) that corresponds to a 6-bit code (64 symbols). Morse coding (1838) also represents a text encoding method, where each alphabetical instant has its own “compressed” exchange value. If one would define the beginning of compression method history, it theoretically indicated any human interaction, where some form of information was shortened to serve a better communication purpose.

The modern concept of compression methodology dates back to the late 1940’s. Data compression methodology and its underlying theory may be viewed as a part of information theory. Information theory defined as a study of efficient coding and also the result that effects the speed and transmission, moreover the probability of error in order to minimise data transmission (Lelewer & Hirschberg, 1987). Shannon (1948) laid down the theory of communication structure between two entities based on strict mathematical equations. The importance of this fundamental work in relation to data compression relies on how written messages (based on alphabet, words, strings etc.) structured and transmitted between two nodes. This surrounding creates the basis to how a given data structure should be modified, compressed or transmitted, yet must remain the same without any data loss.

Fano (1949) published his technical paper about prefix coding based on set of symbol probability. Essentially his algorithm and the Shannon-model gives the same result of text compression, however their approach differ. Yet the world still recall their method as the Shannon-Fano coding.

In the early 1950's David Huffman has published his work (Huffman, 1952) about an optimal lossless data compression algorithm also known as variable-length string coding-scheme that in his time was a very unique approach. Huffman's algorithm still makes the core value of modern block-coding compression methods nowadays.

Around two decades later, in 1977 Abraham Lempel and Jacob Ziv have developed a universal dictionary-based compression algorithm (Ziv & Lempel, 1977) that represents a simplistic, yet very popular lossless compression method based on counting the occurrence and length of each repetitive characters in a given text. Their valuable document became the essential basis of the most used compression algorithm such as zip and gzip. As the types of compression methods appear to be countless and it might be hard to grasp what is the relation between them, Figure 2.1 provides some ideas about their evolution.

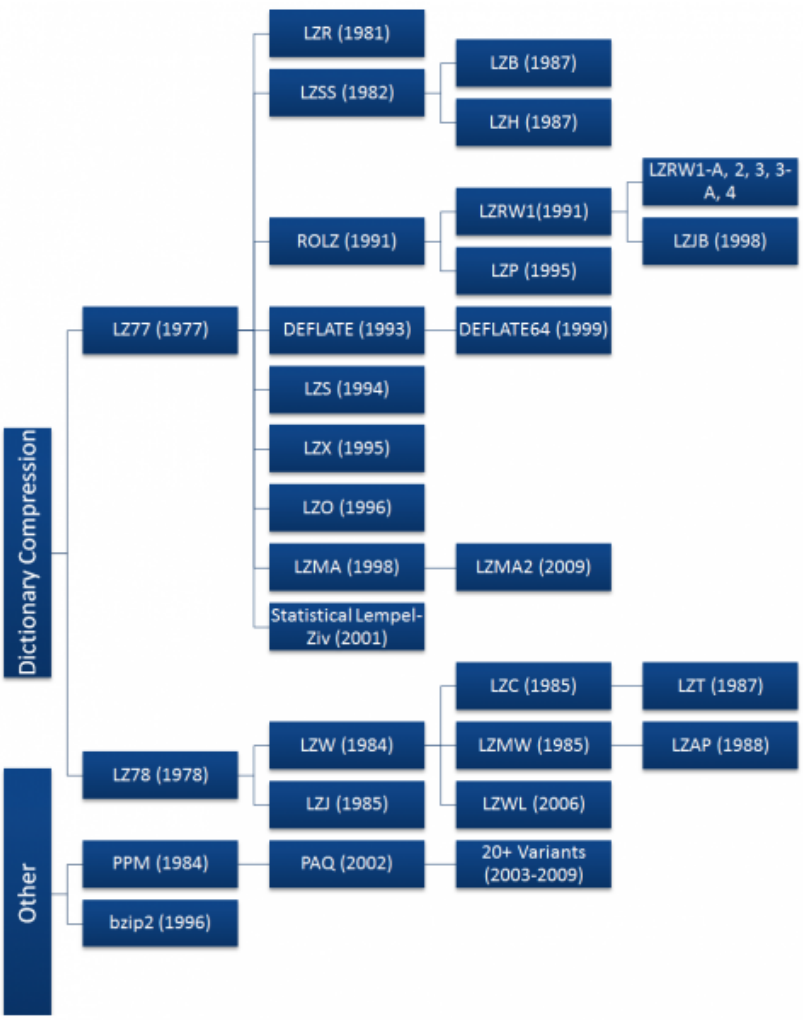

Figure 2.1: Data compression history (ETHW, 2011)

In the mid-1980's, Terry Welch made further progress on the Lempel-Ziv lossless compression (Welch, 1984) and created the LZW (Lempel-Ziv-Welch) algorithm. A distinct difference can be seen this in this period between the application field of the lossless and lossy compression methods. The latter method is utilised by the JPEG (Joint Photographic Experts Group) standard for image compression and MPEG (Moving Picture Experts Group) standard for MP3 or MPEG-4 audio compression. Hosseini (2012) in his journal

article gives a wide review about the lossless compression methods and evaluates the latter mentioned JPEG and MPEG compression techniques as well.

Thom Henderson the head of System Enhancement Associates wrote file compression program (ARC) with the intention to save disk space and released his work in 1985. His compression algorithm became the first that proved the possibility to apply additional compression (Huffman coding and LZW) on compressed files (Mohr & Henderson, 1986).

In the early 1990's Phil Katz developed a compression program as the ancestor of zip compression and after several lawsuit of patent infringements, finally released the pkzip 2.0 and later the pkunzip, both DEFLATE based compression program (Garba & Zirra, 2014).

Rahman and Hamada (2020) analysed the Burrows-Wheeler Transform-based algorithm aka. bzip2 utilising the Huffman coding scheme and concluded that the result demonstrate higher compression rate. Open source compression formats such as the DEFLATE-based zip, bzip, 7-zip formats became popular at this time. 7-zip has strong modularity as it can re-compress additional zip methods as well.

The field of compression remains an interesting developing area for scientists in the future. Context mixing algorithms like PPM (Prediction by Partial Matching) or PAQ (Position Analysis Questionnaire) tends to improved the speed and rate of the above discussed compression method. CRUSH (Abu-Taieh & AlHadid, 2018) can achieve even higher rate of compression in spite its limitation in terms of speed. The Lempel-Ziv Markov (LZMA) algorithm seems to be still a great solution and widely adopted method within many open source compression techniques. An up-coming trend in the past decade by using substring enumeration (Dubé & Beaudoin, 2010) shows reliable result nevertheless.

2.6 Compression Methods Comparison

When it comes to implement any compression method the major question may arise: which compression method is the most suitable to address the specific problem? As several method are mentioned above, it is essential to decide what is the purpose of each procedure and how do they apply in data compression. With other words, what type of data is in our scope and what is the desired goal to achieve through compression. It is obviously rare to choose compression tool based on compression algorithm. However, we should rather take into consideration which algorithm is the most suitable for e.g. text, video, image compression and make our decision accordingly. In order to be able to choose most suitable technique, several survey are offering data compression analysis in terms of speed, latency etc.

Jacob et al. (2012) in their lossless text compression analysis have tested the four most common comparison techniques such as Huffman, Arithmetic, LZ-78 and Golomb by using MATLAB software. They concluded that Golomb is the most suitable for low frequency,

arithmetic coding is for high frequency, LZ-78 for highly correlated data, while Huffman is optimal "*when the probability of each input symbol is a negative power of two.*" (Jacob et al., 2012, p. 20)

Semunigus and Pattanaik (2021) analyses various low bandwidth text, video and image lossless data compression methodologies and categorise which algorithm performs best in each assigned category. Also point out that arithmetic compression types are better with text files and Huffman performs better in big data compression schemes.

Berz et al. (2015) compare four different types of lossless compression algorithm (*compress, gzip, bzip2, xz*) to evaluate the best by speed and compression ratio. The raw data file types scaled from *.png* to OGG sound files through ISO files in various size up to 600 MB. They summarised the result in their technical report. *Compress* is validated as a legacy tool, the zip-types performed outstandingly better, but mainly in text compression, while *xz* delivers the highest compression ratio of all.

Altarawneh and Altarawneh (2011) pointed out in their comparison study undertaken on english text files with different size and texture focused on time ratio, speed, size and entropy containing LZW, Huffman, Fixed-length coding schemes, it is concluded that LZW performed best in all four categories, especially in large file compression. The author also defined above the four sub-category definition (compression size, compression ratio, processing time and speed and entropy) an additional symbolic probability (the probability each character in the uncompressed file) and Hamming weight (number of ones in the N-bits). Latter can be implemented in Hypercube and combined with Huffman can show promising results.

2.7 Hardware Application Field of Compression

Data compression techniques can be implemented on hardware, firmware and software and necessarily involves direct increase of CPU cycles while executing both compression and expansion (Reghbati, 1981). Mittal and Vetter (2015) in their survey have undertaken a qualitative approach to examine the capacity of main and cache-memory system compression on various CPU and GPU architectures with focus on the energy consumption and cost-efficacy of different compression algorithm. The research concluded that due to the bottleneck of compression algorithm integration on certain architectures might require a novel approach combine compression methods and should worth to examine further. Programming multiple-core processor in order to efficiently transfer data between the main memory and the CPU/GPU during parallel DBMS utilisation battles with challenges Besedin et al. (2015) as well. Applying various compression methods increase the speed of data transfer in these scenarios.

Fang et al. (2010) in their journal article propose eight light-weight compression method implemented on different GPU architectures to alleviate performance problem that may occur between the main memory database and the GPU during data transfer.

They concluded that the applied compression not only improves the query processing performance, but reduce the overhead of data transfer by 90% (Nemerson, 2015).

A very promising field where hardware compression technology can be be evolved is CPU hardware acceleration. Intel QuickAssist (Intel QAT) is around for decades and offers integrated chipset-based offloading capability by implementing compression and decompression algorithms to improve the speed of e.g. public key encryption (Intel, 2022).

The work of Ozsoy et al. (2014) presents design improvement by implementing LZSS (Lempel-Ziv-Storer-Szymanski) lossless data compression algorithm on NVIDIA's GPGPU (General Purpose Graphical Processor Unit) versus serial CPU parallelisation. The researcher achieved greater performance with their algorithm on GPU and more effective in data streaming scenarios.

2.8 Computer Security Log File Management

Security log file entries are text-based information collected, stored in a file and contain any information about the state of a network or system. These inputs are the only data that can help to analyse an incident or troubleshoot possible failures. It also helps cyber security or forensics analyst to monitor user activities or characterise and categorise possible events or incidents. The sources location of log files can scale from the physical level of an information system such as servers, routers, switches etc. to the software level such as operating systems, database management system or custom application. Every in-and outbound connection leaves a footprint in a log file. Correspondingly, the preservation of security log files is a must (Ali et al., 2021).

Information Security Regulatory organisations (GDPR, COBIT, SOX etc.) have implemented significant amount of substantial regulation with major focus on application server (mail-server, web server, firewall server, proxy server etc.) log file collection, retention and review. With that said security data has become valuable assets and important source for threat analysis, accessing security posture or comply with audit requirements (Anusooya et al., 2015). The data collected must be accurate, complete and verifiable. From a legislation perspective the collected data must be managed accordingly in case of breach, criminal activities or fraud. To fulfil these requirements the company must save and keep the necessary log files (access log, remote log, security log etc.) on the organisation's server. Using the original log file structure, by time the available storage space will run out. Answering to these challenges log rotation is applied by using data compression techniques.

The US National Institute of Standards and Technology (NIST) defines a set of inevitably useful recommendations in terms of security log file management (Kent & Souppaya, 2006). This standard emphasises the importance of establishing and maintain security management infrastructure, special focus on a robust log management process throughout the organisation's policy in order to meet with data retention requirements and preserve

the data for further investigation to comply especially with the third pillar of the CIA (Confidentiality Integrity Availability) triad. During security audit these important logging files provide evidences whether e.g. electronic communication providers had complied with security regulations. Therefore it is crucial to implement log signing procedure to protect security log files against internal or external attacks (Stathopoulos et al., 2008).

Due to the inconsistent nature of log files timestamps, log contents and formats - including various sources and multiple node points, high quantity data generates on daily basis. As security logs contain all sort of information about the state of the system before, during and after a potential attack, it is recommended to store them on a centralised log file correlation system (Rinnan, 2005) until further observation. Similar approach is recommended by Söderström and Moradian (2013) to securely receive, store and distribute audit log files between different network architectures. By storing large amount of collected log data, one can face to challenges in terms of hardware capacity, that is in a later stage will be cost related. One solution to address this problem to apply compression method considering the type and pattern of log files.

What makes sense of having log file management is greatly summarised at the work of Gorge (2007) where some of the main points outlined such as identifying security incidents, policy violations, fraudulent activity or operational problems within the system. It is advised not to randomly monitor log file entries, instead implement a robust security log file management strategy.

Log file management became a steadily growing business opportunity thanks to strict governance regulation, the continuously increasing trend of cloud log-based management, nevertheless the rise of cyber attacks especially in the energy sector. The capitalisation of global Log Management market has an estimation of 10.52% increasing tendency (Insigthz, 2023) towards the year 2027.

2.9 Security Log File Retention

A critical starting point when managing and archiving compressed security log data e.g. firewall logs, Intrusion Detection System (IDS) or audit log is to comply with certain policies and frameworks. These guidelines not only ensure confidentiality, integrity and availability of these assets, but also set a certain retention time policy. Wilbert and Chen (2012) mention significant breaches that indicated large standards (PCI-DSS, HIPAA etc.) being put in place. In some cases the minimum retention time scales between 30-90 days. Other standards (Leonidas, 2019) regulate the retention time up to 6-9 years.
In the case of security log files, retention period can be between 6-18 months (LogicMonitor, 2022). Akbaş (2023) underscores the critical importance of live log retention policy, recommended to be applied in defence strategy against cyber threats. As the agility of adversaries grows, its is required to build a robust an proactive cybersecurity position and embrace live log retention. The ability to analyse and retain security log files supports organisations to understand the nature of attackers and proactively neutralise their attack

vector at an early stage.

2.10 Compression techniques for security log management

Yao et al. (2022) aimed to provide a bit more technical insight to log management compression, implemented by log management tools such as Splunk[1] or ELK Stack[2]. Both log analyser tools help to monitor and analyse machine-data patterns to improve security and compliance. The authors examine how these system tools utilise small log block compression in prior the large compression process by testing four state-of-the-art methods in real time. They also propose a solution that improves the small block compression rate before the large log block compression takes place. Log data analysis includes data mining and statistics techniques in log file management procedures and it takes place many times in a post-audit manner.

An implementation of real-time log management file compression procedure discussed by Li et al. (2016) provides a feasible and effective audit mechanism using real log cases. The proposed work in a greater grade reduces the processing time of analytical audit and simplifies anomaly detection as well.

Another publication about log file management from Skibinski and Swacha (2007) conducts different compression methods specialised to security log file structures by using the DEFLATE algorithm that compress shorter files then zip. Cloud computing became the backbone for several services and applications and log file management is not an exempt of this. Spillner (2020) in his work presents a comparison model of thirteen different tools for cloud-based log file maintenance. Cloud service providers can further examine and utilise the benefits of compression-related graphs generated by the tested tools.

Log files entry can differ in size and pattern, but contains vital information about anything that is interacted with the system. A novel approach of Marjai et al. (2022) presents how all these information can be preserved by applying template miners and frequency-based algorithms combined with Huffman coding. The achieved compression ratio scaled on an average of 92%.

2.11 Summary

There are various resources presented to create the framework for this thesis. First and foremost the most characteristic books, academic journals and survey were presented to guide the reader to the scientific field of data archiving techniques. One of them includes a wide summary of the major literature are available in this subject. These works form the core of this research.

[1] Splunk - Software for machine generated data analysis and monitoring that provides event management for log files. (Splunk)
[2] ELK Stack - Log management platform that comprises three projects Elasticsearch, Logstash, and Kibana for log file aggregation and analysis purposes. (ELK Stack)

In the next section, the importance of data compression in daily life is outlined and supported by statistical resources to point out the need for data compression techniques. Thereafter a relatively long section assigned to establish the historical context of data compression, including the presentation of different algorithms to answer the upcoming challenges of storing and transferring mass data. It is followed by a bit more technical part with comparative reviews about different compression and decompression methods.

Compression algorithms can serve hardware performance utilisation as well. This part has touched briefly. Finally, the relation between security log file management and compression methods are covered in such manner that the audience can understand what is the significance of security log file compression.

The overall picture shows that plenty of various data are collected and their texture, structure and transit flow are very distinct from each other. Their validation, analysis and further storage possibilities demands also distinct approach correspondingly.

General compression methods like *zip, gzip, bzip2, lz4* and *7z* work in a certain manner and still forms the core of security data log file preservation. Their main characteristics are outlined in the next chapter.

CHAPTER 3

Research Approach

3.1 Introduction

This section aims to outline the testing environment and chosen research methodologies that will serve the artefact for security data file compression. There are five unique compression algorithm (3.2) and their underlying mathematical algorithms are introduced briefly. Many of them are functioning in our daily life without having noticed them. The next section initiates the testing environment (3.3) and justify the reason for preferring these compression methods. A short summary of log file structures and their pattern are presented afterwards (3.4). The testing conditions, implemented tools, testing environment and frameworks are introduced in more depth as well in 3.5. The chapter ends with a short summary (3.6).

3.2 Compression Tool Selection

Lossless compression methods can be found in various operating system and one of the greatest benefit of zip compression family is that they are OS independent. There are five diverse compression tools and their mathematical algorithm are in scope of this study. Each of them utilises a dictionary-type algorithm. Table 3.1 summarises the most important information about them, including only their compression syntax. In case of decompression the syntax are different. A more detailed list of compression synopsis and their corresponding explanation can be found at the end of this section (Table 3.2). These algorithms will be further discussed later in Chapter 5.

Table 3.1: List of test algorithms

Date	Name	File format	Algorithm	Version	Execution path Compression
1989	zip	*.zip*	DEFLATE (LZ77 + Huffman)	11.2.0	zip [OPTION]...[FILE]
1996	bzip2	*.bz2*	RLE, BWT, MTF Huffman	1.0.8	bzip [OPTION]...[FILE]...
1992	gzip	*.gz*	DEFLATE (LZ77 + Huffman)	1.10	gzip [OPTION]... [FILE]...
2011	lz4	*.lz4*	LZ77	1.9.4	lz4 [OPTION]...[FILE]...
1999	7-zip	*.7z*	LZMA	16.02	7z [OPTION]...[FILE]...

3.2.1 Zip

Probably the most well-known open-source dictionary-based compression and decompression algorithm, that is implemented first as a part of PKZIP utility. It consist of different algorithms such as the LZ77 and Huffman coding; together forming the DEFLATE algorithm, defined in [RFC 1951] (Deutsch, 1996). Zip archive can comprise several individually compressed files, therefore it can be modified without applying decompression on the whole archive (Langiu, 2013). Both Windows and Mac OS uses as a standardised tool for file compression. Zip also takes advantage of using Cycle Redundancy Check (CRC) to prevent and detect accidental data change.

3.2.2 Bzip2

Bzip2 compression tool [RFC 5655] (Sourceforge, 2022) originally created by Julian Seward and utilises the Burrows–Wheeler algorithm. As *gzip, bzip2* is rather a single file compressor than a file archiver. This method applies several compression layers, first the Burrows-Wheeler to group together and sort out the similar characters, then Huffman coding to compress the sorted input. Its compression speed and compression ratio is higher then zip or gzip, but as a trade-off the decompression speed is slower (Mark et al., 2009). *Bzip2* uses block-based method to divide the input stream into smaller chunks between 100 kB and 900 kB.

3.2.3 Gzip

Jean-Loup Gailly and Mark Adler developed a free software to provide a replacement for the UNIX *compression* method. The *gzip* compression and decompression tool [RFC 1952] (Gailly, 2023) similarly to zip method, uses DEFLATE. The file format contains a 10-byte header with version and timestamp, in some cases the name of the original file. The body itself holds the DEFLATE-payload, then the footer (8-byte) suppress a CRC-32 checksum with the original uncompressed data (Gopinath & Ravisankar, 2020). Gzip only used for single file compression. In order to compress multiple files, *gzip* must be combined with .tar first, then re-compress again resulted a *.tar.gz* or *.tgz* file format.

The DEFLATE algorithm uses block-based compression, where each block consist of a pair of Huffman code trees and applies the LZ77 algorithm to find duplicates (Oswal et al., 2016).

3.2.4 Lz4

When speed is an important factor, LZ4 compression tool can deliver high speed compression rate (over 500 MB/s) on a single core, at the expense of low compression ratio. Collet (2022) gives more insight how the LZ4 block compression format works in his Github site. This tool utilises the LZ77 algorithm-family without any additional combination of other compression algorithm. Because of its high speed performance it is a subject of a hardware acceleration study by J. Kim and Cho (2019) with a result of 3.8 Gps compression throughput. In English text environment LZ4 combined with Huffman proposed by Tariq et al. (2023) achieves an average of 73-78 % compression rate.

3.2.5 7-Zip

The 7-zip compression format (Pavlov, 2024) originally created for file archiving and data compression, however it supports file encryption (AES-256) and algorithm pre-processing as well. It belongs under the GNU Lesser General Public License and utilises the LZMA algorithm. 7z can be inappropriate for backup archiving as it does not store file system information such as file permission, last modified date or creation date. Therefore one workaround is to compress the given data with *tar*, then apply *7z* (Gastegger, 2020).

Table 3.2: Compression methods with syntax

Command	Synopsis	Description (performance related)	Comments	Help / CLI manual
zip	-0 -6 -9	*Lowest compression* *Default value* *Highest compression*	Min. compression level, max. speed. Max. compr. level, slow execution.	zip -so man zip
gzip	-1 or –fast -6 -9 or –best	*Fast compression* *Default value* *Best compression*	Fastest with min. compr. ratio. Slowest with max. compression ratio.	gzip –help man gzip
bzip2	-1 or –fast -6 -9 or –best -s	*Set block size to 100 kb* *Default value* *Set block size to 900 kb* *Uses less memory*	Fastest, least compr. level. Best compression, slow speed. Memory consumption at most 2500 kb	bzip2 –help man bzip2
7z	-mx=0 -mx=1 -mx=3 -mx=5 -mx=7 -mx=9	*No compression* *Low compression* *Fast compression* *Default value* *Maximum compression* *Ultra compression*	Doesn't compress at all, copy mode. This is called fastest mode. Sets various parameters automatically. Maximum compression level. Fastest option.	7z –help man 7z
lz4	-1 or –fast -9 -12 or –best	*Default value* *High compression* *Highest compression*	Max. compr. ratio, speed irrelevant. Highest compression level.	man lz4

3.3 Test Plan

The purpose of this experimental testing plan is to introduce the environment where the research will take place, the data structure that serves as the subject of the compression algorithm and finally the the compression tools that are going to be executed. The goal of the testing procedure is to closer examine the performance of each compression algorithm under the very same circumstance and record their performance. After their data performances are captured and collected, we are going to visualise these results and draw the appropriate conclusion: which compression tools performs the best in terms of speed, compression ratio in regards to various files and which compression method is the more suitable to compress and decompress certain types of security log file.

3.4 The Test Environment

The following hardware and software specification are in present during the entire testing procedure.
Host machine configuration:

- AMD Ryzen 7 7735HS 3.2 GHz, 8 cores, 16 logical processors
- 32 GB DDR5 4800 MHz
- Micron 2400 SSD 500 GB
- AMD Radeon RX 7600XS
- Windows 11 Home 64-bit version 10.0.22631

Client machine configuration:

- VMware® Workstation 17.0.0 Pro virtual machine
- 8 cores CPU
- 16 GB RAM
- 100 GB HDD
- Clean installed Ubuntu Server 22.04.4 LTS

All tests are repeated 15 times. The method for processing is to be discussed in next chapter.

3.4.1 The Data Sets

It is previously discussed what types of data can be archived by using lossless compression algorithm. The diversity of data gives us a wide range of compression techniques to choose from. At the very first stage, we prepare the raw compression data aka. corpus that contains diverse log files, harvested from servers, IDS logs, packet capture etc. Their size have a variety of 100-250 MB. This proportion seems reasonably large enough to deal with and contains thousands of lines of information with different textual structure. As the research subject is limited to work with text files; image, audio or video files are falling out of scope. The following security log data types however are in scope:

- Syslog [RFT3164 / RFC 5424]: plain text that includes hostnames, process IDs, log messages, timestamps. Regular text formats: .txt, .log .docx, .json
- Common Event Format (CEF): contains .csv files.
- JSON (Java Script Object Notation): contains plain text file with .json extension.
- CSV (Comma-Separated Values): contains tabular data in plain text in with .csv extension.
- XML (Extensible Markup Language): contains plain text elements in structured form with .xml extension.
- Apache HTTP Server Logs (Common or Combined Log Format): contains access end error logs in CLF file format
- NCSA Common Web Log Format: used for recording HTTP server activity, contains plain text in structured format.
- Packet Capture (PCAP): sotres network traffic in .pcap or .cap format, contains following information: IP source and destination address, timestamp, file size, protocol name, port numbers etc.
- Firewall log, FTP (File Transfer Protocol) server log, SSH (Secure Shell) log, Nginx Access log: .log extension with log information in plain text format.
- Emails: .eml, .pst, .mbox, .mime, .edb files contains messages between client and server, mailbox data, includes text-based files and attachments.

These files are represent various state of system or network security. These are the main and almost only source of valuable information that can be initialise in case of investigating a security incident. They can also be useful to monitor or debug any error occurs during system lifetime. Syslog as a standard logging protocol provides vital information about system activities or potential security breaches. Especially server logs can comprise crucial information about the state of an adversary during a potential computer attack. An analysis of Pcap files contain raw network data between two node points can reveal network-based threats, malware or suspicious activities. Firewall logs can provide records about IP addresses that are blocked or unauthorised. SSH log files can detect brute-force attacks or unauthorised access to the system. Overall these log files register security events across different layers of the computer and network architecture. Collecting, storing and analysing them helps to enhance the overall security posture and mitigate potential risk to the any organisation.

3.5 The Testing Framework

The framework for security log data compression-algorithm testing outlines how the testing procedure builds up, including all sub-tasks and their relation to each other. It also gives a brief insight of the general execution flow within the framework. Each and every phase will be described, giving a grounding explanation how and why they are connected together. A part of the testing framework script automation is initialised to enable repetitive task execution without human intervention. The reason for implementing automatised solution is to receive consistent, accurate data results and measure the productivity of the outlined

framework. Automated framework in this mean facilitates scalability and replication of various task across multiple working phases.

3.5.1 BASH Script Execution Flow

To only compress data one can run a CLI command with the required compression method by specifying the archive type, its arguments and the source and destination path. As a result, the raw data is compressed, the execution is finished. In our case we aim to collect and analyse relatively large amount of statistical data during the whole execution procedure to draw performance graphs of each tool. This approach is time consuming and requires some sort of automation to run. The logic of execution of each compression tool is not complicated, but the amount of execution in terms of each an every algorithm, including their different synopsis demands more than a manual approach.
To answer this challenge a BASH (Bourne-Again Shell) script (Appendix A) is developed that contains the following execution phases:

1. Defining the input file and its location - optionally pass as a CLI argument. Define the name of the output file.
2. Defining the set of test to run for each compression tools.
3. Defining the array of compression methods, containing their names.
4. Utilising the *main for loop* that iterates through each compression tool command is described in phase 3.
5. The *main loop* maps the dataset and runs thorough each element.
6. The *first nested loop* within the *main loop* specifies the number of rounds (15) for each compression level block to run.
7. The *second nested loop* within the *first nested loop* defines each compression level (1-9) and creates separate compressed file for each level.
8. The program jumps back to the *first nested loop* and execute the decompression command and creates decompressed files for verification.
9. Defining compression and decompression syntax for each tool.
10. Recording the execution time (speed) for compression and decompression seconds.
11. Recording the compression ratio by dividing the uncompressed size with the compressed size.
12. Calculating the average speed and compression ratio.
13. Printing out the results for each as STDOUT as well as append it to an identical file.
14. Creating a summarised file with all the recorded metrics.

The multiple tests are important for statistical validity. The script begins to identify the input parameters (path to file, compression tools, output file) and loops through each of them. A nested loop in within runs the specified test. The second loop aims to iterate through the compression tools and execute them, also records their speed and compression ratio. After the last round the script count the average speed and ratio. As a final step, the script append the results into a specified file and prints the result out on screen. This approach so called data mesh will ensure to collect data on a centralised manner to further analysis.

BASH scripting is used a default command line and script language of the UNIX operating system, additionally available on nearly every modern operating systems. It's simplicity and effective way to automate repetitive task makes it a good candidate to answer our challenges. By applying BASH scripting the researcher can run multiple commands on the same time, do interactive debugging and automate tasks (Ebrahim & Mallett, 2018).

3.5.2 General Flow of Execution

In the preparation phase the testing environment (virtual machine) is configured and a separated virtual drive is assigned to keep the extracted data safe in case of any failure occurs during the execution. One advantage of using virtual environment that the state of the client machine can roll back at any point of time to a previous state. This provides a secure environment for testing. All the reading, writing permission are given to the administrator and the security data corpus is located. The required parameters such as file size, content type and compression level are decided. It is planned to run every compression algorithm 10, preferable 20 times to get more valuable and stable results. The fastest and slowest recorded data in each phase are eliminated.

The compression starting phase begins with running the BASH script to generate test archives, while recording the execution time and the average. It is possible to set up further metrics as well according to our needs. The integrity of the extracted files will be verified right after the archive extraction is finished, same time with error checking.

The collected data, using the appended file will be processed. The calculated average and the standard deviation for metrics are defined. To measure the performance of the diverse synopsis of all algorithms, we run a statistical test. Having the final results, graphs are generated to visualise the performance and derive conclusion to the assigned question.

3.5.3 Measuring and Metrics

There are metrics that forms the base of this research:

- **Compressed size in MB**: the size of the compressed file in human-readable format.
- **Compression rate**: the proportion of compressed size to uncompressed size.
- **Compression time**: time spent to compress the original file in seconds.
- **Compression speed**: bytes/second rate.
- **Decompression time**: time in seconds used to extract the file.
- **Entropy**: the unpredictability of the data pattern.

3.5.4 Expected Results

The following results are in interest from the perspective of the research:

1. Compression speed and compression ratio of the different synopsis.
2. Average compression speed and average compression ratio accordingly.

3. Compressed and decompressed file entropy
4. Diversity across multiple runs.
5. Recorded statistical data and their divergence between each synopsis and between the different algorithms.

3.5.5 Processing and Analysing the Results

The finalised research data will be further processed to calculate the standard deviation. Average metrics are created throughout the various runs. Statistical data collected and processed to check if there is a significant contrast between the collected data groups. Implement charts and graphs to visualise the results for easier interpretation. It is also intended to identify the fastest and the most efficient compression algorithm in accordance to different file size. It needs to be determined what are the trade-offs between each tool's synopsis and their relation to one another as well as between the compression ratio and speed. In case of any unexpected result may arise during the period of research, it is necessary to further investigate them.

3.6 Summary

The researcher aimed to put into context an archive texting framework by evaluating their performance. In order to do that a brief introduction to each element is presented. At this stage deeper analysis is not necessary. However it is important to understand the structure of the research, realise the function of each tools being tested and initiate a systematic approach how the performance of these compression techniques are recorded, analysed and presented. The test environment and the underlying hardware components are outlined. Information is given about the raw data formats and their importance and relevance in relation to the study. The function and execution flow of the automation script is also presented. The collected data analysis involves processing the data, calculating the speed and compression ratio and the standard deviation accordingly between each algorithm. Deciding about the best and fastest compression algorithm is the subject of the future testing procedure that is going to be presented in the next chapter.

CHAPTER 4

Data Compression Evaluation

4.1 Introduction

Working with various types of security data might be challenging. For the first it raises questions in relation to the CIA (Confidentiality Integrity Availability) triad. Especially the second pillar, integrity that is to be concerned when it comes to start working with security related dataset. For instance network traffic capture files are direct sources to malicious activities and can contain different types of malware. Therefore these resources needs to be approached with due diligence. From our perspective it is essential to check the integrity of all files prior any examination takes place even though the the files are filtered and refined before making it available for public use. To ensure the integrity of the downloaded files each file were validated by applying SHA256 algorithm.

In some cases there was indeed challenging to retrieve huge amount of diverse security data to work with. One reason behind this is that security data is confidential and hard to be acquired. It contains sensitive information about different states of a system or network, that should be handled with proper secrecy. In other cases it was challenging to find reliable sources that might provide all sort of files that falls under the categories of this experiment. Gathering diverse security log file data allows us to observe how each compression method performs in different situations. Having a wide range of resources is essential for thorough research, helping us understand various logging processes and compare different compression algorithms. The differences in security log file structures affect how compression algorithms are applied, shaping their effectiveness in different scenarios.

Each log dataset is constructed to follow a systematical process to categorise and preserve security log events. This can determine what compression algorithm would be the most suitable being applied in different scenarios accordingly. Each dataset contains unique

information that differentiate one dataset from another. These information are arranged in a certain pattern to represent the data in a clear and practical manner. The next section are going to provide more background information about how each security log file collected for compression sequence analysis.

This chapter serves as an introduction to the raw data set, the focal point of this study. It aims to supply in depth information of the security data collected. It also describes their structure and emphasis their function in the field of computer security.

4.2 Security Log Corpus

A log file is a contextualised form of event or incident that has happened in a specific time during system runtime. Security event and security incident are observable records that provide historical and measurable information about what and when happened. It can contain errors, sign of intrusion, connection information etc. These are transmitted in several forms. A log file can be found in structured, semi-structured and unstructured format (Table 4.1) described by Sharif (2022).

Table 4.1: Log data structure

Unstructured	Semi-structured	Structured
Plain text in different forms	Both structured and unstructured	Log or JSON format
Human-readable	Human-readable	Less human-readable
Hard to parse	Complex parsing logic	Easy parsing
Manual interpretation required	Half automation possibility	Automated
Not suitable for security logging	Challenging to work with	Fast troubleshooting capability

The unstructured format mostly contains unorganised log data in a form of text that however shares information about the event occurred but lack of consistent and sufficient context. Semi-structured logging is half way to be organised, but often lack of recognisable format. This means that the timestamp and user information are at place but the message is somehow vogue and embeds all sort of contextual details that makes the whole text incoherent. However a structured logging is more consistent as each important component adheres to a recognisable format. Based on this information security log data can be the main source of investigating a security incident or problem-shooting a network malfunction. The basic structure of a security log file includes at a minimum of the following:

- Timestamp: accurate time and date of the event logged in the system. It can serve as a reference point for user to track the sequence of events. They are often formatted by using standards like ISO 8601 (YYYY-MM-DD HH:MM:SS) for higher consistency.
- Event type/ID: Log entries contains the types of the event recorded and marked e.g. logging attempt, file access, startup, but mostly parsed with a unique identifier.
- User information: It includes username or user ID, session ID or any distinctive identifier that differentiate one user from another. This information helps to identify potential anomalies that are not the part of a general function.

- Source: logged information about the origin of the event such as IP address, hostname, device ID etc. These helps to locate where the event occurred.
- Description: provides additional information what exactly happened marked e.g. error, unauthorised access etc.
- Severity level: defines the gravity of the event and helps to categorise them accordingly: warning, alert, error etc.
- Status: simply describes if an event resulted in any action such as success, failure, denied access etc.

Security log file data are text-based files and their structure consist of characters used in the English alphabet. An English-based standard character set for security logging commonly uses ASCII (American Standard Code for Information Interchange) or UTF-8 (Unicode Transformation Format-8). In correspondence to that these are the most commonly used character sets in security log data:

1. Uppercase letters: A-Z (26 characters)
2. Lowercase letters: a-z (26 characters)
3. Digits: 0-9 (10 characters)
4. Punctuation marks and symbols: `!¨#$%&'()*+,-./:;<=>?@[ ]_`{ | }~^` (33 characters)
5. Control characters: non-printable characters used for control functions such as tab, carriage return, line feed etc.
6. Space character: ' ' (1 character)

This fact also determines what possible ASCII character set (in total 128 characters) can be appear in a log file and what is the frequency and probability of the occurrence of each alphabetical character. This basic knowledge comes useful when we deep dive into the function of compression algorithms in the next chapter.

The available sample files ranged from few hundred kilobytes up to hundreds of gigabytes. Especially .cap and .pcap files can be tremendous in size. Hence some adjustment needed to be taken along the way to optimise the size of the corpus. The structure of security log data files are homogeneous and they follow certain patterns to make it easier to work with. Some sort of adjustments are implemented to reduce the size of the corpus in order to enhance the speed of the work processes. As a result to this a maximised size of ca. 100 MB data are created for each dataset corresponding to a total of approx. 1 GB of log file data. This might not appear a significant amount nowadays, but if we want to count how many lines of information they are including, we have to consider millions of lines. For instance a 100 MB SNORT log contains up to 2,304,619 lines of information. The next couple of section are going to introduce the security data corpus. Each corpus contains a description and a two lines actual information retrieved from the test server by using *file* and *ls -lah* CLI commands to verify the type and size of the original files.

4.2.1 Access Log Dataset

System logs consist of system generated messages of the computer OS, network device or application. These logs are created for auditing, troubleshooting or monitoring purposes and provides valuable information about the state of the computer systems or network devices. System logs notify the administrator about the health of the system hardware and software.This corpus is retrieved from Sconzo (2024) and marked as a Squid Access log that is a combination of several sources with an original size of 210 MB uncompressed. It mainly contains web-based access requests and firewall logs from an Apache web server.

accesslog.log	
File type	accesslog.log: ASCII text, with very long lines (439)
Original file info	-rw-rw-r– 1 tamas tamas 210M Dec 31 2014 access.log
SHA256 value	19301e16848c39b83eaae9e2f6af79a119e6b14eb756dfdd3802d27b21ff3d48
Entropy	5.381976 bits per byte

4.2.2 Apache HTTP Log Dataset

Apache HTTP server logs are files generated in a Linux-based environment and record information about client request, server response and other web-related activities between the server and the client. These logs provide information about web server activities especial focus on web traffic. This sample is taken from Security Repo, generated and updated by the community (Chuvakin, 2009) and also serves educational purposes.

apachelog.log	
File type	apachelog.log: ASCII text
Original file info	-rw-r–r– 1 tamas tamas 320M Mar 10 13:37 2006-allog.6
SHA256 value	e398ef4f0f7ae8f97f11449336bbeb4e245351b5e14e7c4064b6284af364bdef
Entropy	5.395532 bits per byte

4.2.3 Connection Log Dataset

This corpus represent connection attempts to Apache web server (Sconzo, 2015) that are resulted in different status. Connection log files record established server-client connection and commonly used for network monitoring. This includes but not limited to analyse network performance, troubleshoot connectivity issues and identify potential security threats. Security Data Analysis Labs established to provide free, open source educational material and workshops. This corpus contains 22 million flow events with a size of 2.6G uncompressed.

connectionlog	
File type	connectionlog: ASCII text
Original file info	-rw-r–r– 1 tamas tamas 2.6G Aug 31 2014 conn.log
SHA256 value	47cd565b5e46c7ff0b535b00292b39f49dcb7438763a43fbf00474e68bfcf726
Entropy	4.712926 bits per byte

4.2.4 Csvhoneypots Dataset

USB-IDS-1 dataset (de Sannio, 2022) provides large amount of filtered *.csv* dataset of labelled network flows. The network traffic obtained through CICFlowMeter after the *.pcap* files were distilled. Csv files are containing comma separated values that are basically structured data, captured by honeypot systems. Honeypots are decoy systems that attract malicious actors and monitor them to further analyse their behaviour. This corpus mostly comprises huge volume of numbers that are recorded to summarise data information of a DDoS attack named Hulk. All HTTP connection request that purpose were to overwhelm the server are recorded and stored in *.csv* format. This file type often used to provide a more concise and structured layout of log events to be further processed and categorised. It has lower entropy as the previous file samples, that indicates a more predictable structure.

csvhoneypots	
File type	csvhoneypots: ASCII text, with very long lines (670)
Original file info	-rw-rw-r– 1 tamas tamas 461M Jan 26 2022 Hulk-Evasive.csv
SHA256 value	23c0b3ac02e842a09697bb066827f2d300d412b13b7e532197049af99c2b7a9e
Entropy	3.758907 bits per byte

4.2.5 Email Dataset

Nielsen (2023) provides a set of fraud email in *.csv* format derived from the original datasets. This phishing email curated dataset spans between 1998-2022 and contains various types of legitimate phishing and fraud emails. Email logs are collections of recorded communication between parties. This log contains email metadata structured as sender, recipient, address, timestamp, size also holds other relevant information within the body of the email structure. These messages are valuable resources for security analyst and are helping to identify and analyse spamming, phishing activities as well as security incidents. Phishing emails nowadays are very popular tool in the hand of malicious actors and serves key component during the deliver phase of the Cyber Kill Chain. Analysing these crafted emails offers more insight to realise what are the motives behind phishing campaigns. It is essential to obtain these messages for further analysis and due to the enormous amount of email-exchange that generates thousands of mail on a daily basis, it may be crucial to preserve them for further investigation.

phishingemail.csv	
File type	phishingemail.csv: CSV text
Original file info	-rw-rw-r– 1 tamas tamas 12M Apr 4 19:39 nazario5.csv -rw-rw-r– 1 tamas tamas 7.5M Apr 4 19:37 nazario.csv -rw-rw-r– 1 tamas tamas 18M Apr 4 19:40 nigerian5.csv -rw-rw-r– 1 tamas tamas 8.8M Apr 4 19:41 nigerianfraud.csv -rw-rw-r– 1 tamas tamas 15M Apr 4 19:42 spamassasin.csv -rw-rw-r– 1 tamas tamas 40M Apr 4 21:06 trec6.csv
SHA256 value	afef69828165619ced6661ca14a95444a1db82c7b25ae9c2619bbbb108844a44 b8fbc4158fbdfaff1ed98584c43d72e283c6352d7ade4d457d34c1d79488d184 3171eeb9bf560059e7f88458efde65c056fc939b842a5edc158f2d7bb887c8b7 f5d6930763cae6feeae6484131c79655a7440abbb97d6f51eab757aea4e6ab4b 3bfe8f8abff89f69a98456be50413c2fcb20a476141116dd84ee1507b980c00e 8ca6b1249f30f0a74790b8a34d4dbe248d526cfb0b13227b41970a1ebd638be5
Entropy	5.397640 bits per byte

4.2.6 Files Log Dataset

This corpus technically was a part the National CyberWatch Mid-Atlantic Collegiate Cyber Defense Competition (MACCDC) and represents network related traffic captured and filtered down by Mike Sconzo (Sconzo, 2024) to contribute to the open source community.

fileslog.log	
File type	fileslog.log: CSV text
Original file info	-rw-rw-r– 1 tamas tamas 178M Sep 21 2014 files.log
SHA256 value	9b18d62a9e43d47d1713e47c5c1668e7d28f0bad7e2f2748ba51631bd414377c
Entropy	5.040807 bits per byte

4.2.7 Honeypots JSON Dataset

There are several advantages of using JSON (JavaScript Object Notation) in log management. It is relatively easy for human to read and write the code, likewise easy for machine to parse. The key-value pairs (objects) allows the system to create well-structured synopsis over the logging processes. Modern logging systems utilise the benefits of the JSON coding language to create and distribute log files in a lightweight style. JSON parsing structure can be implemented in a wide variety of programming language platforms. The JSON corpus includes quite a rich data log that comprises almost 1 million lines. This dataset is originated from SecRepo and contains various honeypots (Amun and Glastopf) that is used for BSides presentations (Sconzo, 2024).

honeypots.json	
File type	honeypots.json: JSON data
Original file info	-rw-r–r– 1 tamas tamas 427M Feb 28 2017 honeypot.json
SHA256 value	df61f09c8de8a04cdb5dc7d8b5e734e37fd951b17c63f1a3fb0bf7b54fbd03ed
Entropy	5.018197 bits per byte

4.2.8 HTTP Log Dataset

This HTTP dataset (Giménez et al., 2010) represent thousands of HTTP GET and POST request to a web server and corresponding responses. It comprises IP addresses, hash

values, URL etc. Each line represent a single entry that contains space separated fields with numeric and alphabetical values starts with the timestamp and ends with the HTTP status code. The given dataset contains base functionality for HTTP log analysis. This particular file has significantly lower entropy, which makes

httplog.log	
File type	httplog.log: ASCII text, with very long lines (378)
Original file info	-rw-rw-r– 1 tamas tamas 1.3G Mar 11 22:28 http.log
SHA256 value	d5a5327e55f418754e139b0faf8ab0b1fbbc241142f55e708766d4ff0bfd7c0b
Entropy	4.752814 bits per byte

4.2.9 Network Traffic Dataset

A network traffic log generally contains all the communication and logged activities captures between network devices e.g. routers, switches, firewalls and supply a detailed information about the network packet flow. It has a chronological set of records that comprises the source and destination IP address, protocol, packet size, timestamps of the data packet. Their purpose is to monitor network traffic, analyse network behaviour or detect anomalies. Packet capturing helps to identify and solve network performance issues, detect packet loss or network missuses. The process so called packet sniffing enables the user to gain more insight about the nature of traffic occurs inside the network. Pcap files are very often include packet-level evidence that can further help to reveal the real cause of network misbehaviour. This set originated from a japanese research working group called Mawi (Measurement and Analysis on the WIDE Internet) that carries out network traffic measurement, analysis and evaluation as the part of the WIDE (Widely Integrated Distributed Environment) project. The latter collaborates globally with universities and research institution to develop and operate education environments. This open source database helps to assist researcher to detect network traffic anomalies. The database is updated in a daily manner, uses tcpdump tool to dump the traffic and tcppdpriv to eliminate confidential information of the trace files (Cho, 2000).

mawitraffic.pcap	
File type	mawitraffic.pcap: pcap capture file, microsecond ts (little-endian) version 2.4 (Ethernet, capture length 96)
Original file info	-rw-rw-r– 1 tamas tamas 13G Mar 21 14:12 202403211400.pcap
SHA256 value	54108a06e4255a7066cf8d1910343d61cf661747cc2203b69a2fe6240ead06f8
Entropy	6.683626 bits per byte

4.2.10 SNORT Log Dataset

SNORT is an open source IPS (Intrusion Prevention System) and IDS (Intrusion Detection System) for detecting and defining malicious network traffic, maintained by Cisco. It has an function to perform real-time traffic analysis and packet logging using IP protocol networks. SNORT has an ability to detect port scans, buffer overflows, SMB (Server Message Block) probes etc. and has function mode to log the captured packets on disk in different file format. In our case we are going to use *.pcap* raw file data (WRCCDC, 2021)

originated from the WRCCDC archive, that is a publicly available source for students and professionals on a completely static NGINX server. The traffic monitored by GoAccess real-time web log analyser tool. The network traffic grouped by date and distributed in smaller (appr. 500 MB sizes) compressed chunks to support an easier work flow. If we have a look at the entropy value, this files sample has exceptionally higher value that indicates that its structure less predictable than the other file samples.

snortlog.pcap	
File type	wrccdc.2024-02-17.085358.pcap: pcap capture file, microsecond ts (little-endian) version 2.4 (Ethernet, capture length 262144)
Original file info	-rw-rw-r– 1 tamas tamas 477M Feb 18 07:07 wrccdc.2024-02-17.085358.pcap
SHA256 value	dee5771e8b4a1e9eb11b9cc28c132718d243f83a329b841d4fbb70a6d7793516
Entropy	7.897905 bits per byte

The following Table 4.2 represents a summary of each data set that is in focus of our interest during our analysis. It essentially contains information about the source files, also describes the structure of the trimmed files, their size and the data they contain. The date of the files represents almost two decades of logging history, yet their structure remained almost the same.

Table 4.2: Security log corpus

Name	Date	Source	Original data size	File name	Reduced Size	Contains	Number of lines
Squid Access log	2014	SecRepo	210 MB	accesslog.log	100 MB	IP address, HTTP request TCP	771,035
Weblogs - Security Repo	2006	SecRepo	320 MB	apachelog.log	100 MB	Src. IP, Dest. IP LEN, TTL ICMP, SSHD	567,621
Connection Log	2014	Security Data Analysis Labs	2.6 GB	connectionlog	100 MB	TCP, HTTP MD5, IP	868,556
Csvhoneypots dataset	2022	USB-IDS-1	461 MB	csvhoneypots	100 MB	IP, time, date	163,979
Phishing Email Curated Datasets TREC_05.csv	2005-2007	Zenodo TREC public corpus	118 MB	phishingemail.csv	100 MB	URL, sender name receiver name, subject date, time	1,126,834
Files log dataset	2012	SecRepo	178 MB	fileslog.log	100 MB	Time, MD5, SHA1 HTTP	470,948
Honeypots JSON dataset	2017	SecRepo	427 MB	honeypots.json	100 MB	Time, date, port nr. HTTP request, MD5	994,692
HTTP log dataset	2012	SecRepo	1.3 GB	httplog.log	100 MB	HTTP GET, HTTP POST time, date, URL, IP	352,113
Network traffic dataset	2024	MAWI Working Group Traffic Archive	13 GB	mawitraffic.pcap	96 MB	Time, date, source IP destination IP, length, port	502,128
SNORT log dataset	2021	WRCCDC	477 MB	snortlog.pcap	100 MB	Event ID, SHA value, ProcessID, UserID	361,972

4.3 Summary

There are ten different types of security log data are introduced in this chapter. The researcher tried to collect the widest variation of log files in terms of structure, size and logging methods to provide a solid basis for a reliable statistical analysis. The variety of data scales from ASCII text files through *.csv* files to *.pcap* files. Each type of files are validated with SHA256 algorithm and their entropy are measured. It is also revealed that however log files are human-readable type of information sources, their structure and application field differs. The more structured the data, the higher the chance to maintain them properly during analysis. The form of log data structure also defines what are the most suitable compression algorithm for certain log files. In addition to this, there certain

log pattern characteristics and similarities are also defined in this section. Hereby needs to be noticed that, however there is no significant difference in term of file size between each file sample, yet *mawitraffic.pcap* deviates by 4 MB that had to be scaled up to perceive a consequent calculation and with that standardise the file set equal to 100 MB each. The next chapter aim to discuss the process of analysis on the previously presented dataset.

CHAPTER 5

Discussions

5.1 Introduction

This section presents the evaluation process to further investigate five different compression tools *zip, gzip, bzip2, lz4* and *7z* by evaluating their performance on the security log data corpus provided in the previous chapter. After a short overview (5.2) that includes interesting metrics about the overall performance of the entire project, the preparation phase is outlined (5.3). There are several measurement criteria falls into our scope, very often they are coupled to have more in depth understanding about the overall performance of each compression algorithm. These measurement categories are clearly explained under section 5.4 a dedicated subsection. The main concern when measuring the performance are the following factors: compression time, compression rate, decompression time and entropy. Additionally compressed file size is also recorded to provide more user friendly information instead of rigid metrics. The efficiency of compression tools very much depends on the structure of the input file and the characteristic of the compression method. The latter is defined as lossless compression, while the raw data represents different structure of text files with the same size. There are multiple visualisation techniques utilised in this work, more details about these can be found under section 5.5.

After the input and output data are cleaned and consolidated each compression method (5.6, 5.7, 5.8, 5.9, 5.10) is evaluated. The evaluation process including to calculate the entropy, byte frequency, measure compression ratio, the overall average and elapsed time. Additionally, average and total average of each and all process are calculated as well. The generated research data not in all cases serve with valuable statistics for our purpose, so these records are filtered and only applied when they were applicable. This section also aims to compare and contrast our findings with other literature that are available in this field. Tables and graphs are drawn to help visualising the performance of each compression

utility to get an overall picture (5.11) as well as to provide recommendation (5.11.4) in accordance to different use cases.

5.2 Experiment Overview

Ten different security log data files are tested with five compression methods. To automate and record statistical data, 5 different BASH script are implemented. Each script generated 9 compressed file (level 1-9) for each file type and 2 statistical data file. In addition to this 1 file is also written with the total results. The decompression script produced 1 output file to cross-check with the original file. There are 122 files are produced in case of each compression algorithm in total. This sums up to an overall 610 files on each round where all the five script are executed. This is repeated 4 times, in two occasion the 5 script ran parallel to verify if the given results significantly differ. All together 2,440 files are created occupying approx. 44 GB of disk space, whereas the overall size of filtered raw data is 1 GB. The consolidated master data for visualisation contains 6,750 lines and 31 columns. The entire test process took approx. 2-3 weeks.

5.3 Preparation

The deployed virtual server (Ubuntu Server 22.04) provided a solid and scalable research environment to examine the compression tools. The zip and lz4 compression programs however was not the part of the default package and needed to be installed manually. The virtual hard drive set to 100 GB which is suspected to be enough regarding file compression usually aims to consume less space. Yet downloading and cleaning raw data required around 50 GB free space after server deployment. This amount of used storage space is accumulated along several phases. Occasionally the entire test had to be restarted from zero to scale up the performance of each compression algorithm and to fine scale the automation script. Between host (Win 11) and server SSH (Secure Shell) connection is established for code execution using the Linux command line and BASH program language. The gathered language of the text is based and limited on the English alphabet including special character set as well. The structure and content of files differs, but their structures show similarities.
The original size of *mawitraffic.pcap* was smaller (96 MB) compared to the other files analysed in this report. To ensure comparability, a proportional rate was calculated using a mathematical formula: Test result / (Original size / 100). This adjustment allows for meaningful comparisons between the data from this file and the other test data included in the report.

5.4 Units and Measurements

5.4.1 Compressed Size

Compressed size gives the result after the original file undergone a given compression process. The process involves to decode the content of the raw file and shrink its size by eliminating its redundancy resulting a smaller size. The size is generally measured in bytes, in this case it is converted to MB to provide a more understandable unit. Smaller file size can lead to faster file transmission and lower network bandwidth usage.

5.4.2 Compression Rate

Compression Rate is a metric to calculate the effectiveness of the compression algorithm and represents the ratio of the uncompressed file size to the compressed file size. It can be defined with the following mathematical equation:

$$CompressionRatio = \frac{UncompressedFileSize}{CompressedFileSizeTime}$$

The higher the number of compression ratio, the more effective the compression algorithm is. Conversely, a lower value indicates that the compression algorithm achieved less reduction is file size. If the original file size is e.g. 100 MB, as in our case and the compressed file size is 20 MB, this shows a rate of 5 and indicates 5:1 compression ratio. This consequently means that the file has been compressed to 1/5th of its original size. The compression ratio naturally depends on some factors such as the type and structure of the compressed file, the applied compression algorithm used, and the parameters given during compression code execution. Different compression tools might achieve different results applied on the same file.

5.4.3 Average Compression Ratio

To measure how effective the compression ratio is on average the following formula is used:

$$AverageCompressionRatio = \frac{\sum OriginalFileSize}{\sum CompressedFileSize} \times \frac{1}{Numberof Files}$$

Here the first part of the sum of the original uncompressed file size is divided by the sum of all compressed files times the total number of files that are compressed. This measurement type provides an overall average indicator of how much all the files are compressed on average. A higher number provides higher efficiency, a lower number indicates how inefficient the compression methods is. This measurement comes handy when we want to compare all the five tools to see their overall performance on all files.

5.4.4 Compression Time (s)

Compression time refers to the duration it takes to compress the raw file data. It is typically measured in seconds and represents the time elapsed between the starting phase and the end when the compression process is completed. The time of compression can be very much depend on the following factors:

- **Data size**: the larger the dataset the longer it takes to compress. This is due to the fact that the amount of data per bytes and its processing speed is directly proportional.
- **Compression algorithm complexity**: the higher the algorithm complexity the longer it takes to achieve the compression. Here the time is complexity level determines the duration.
- **Computational resources**: the performance of the CPU and RAM can significantly impact the compression time. In our case this is steadily constant.
- **Compression parameters**: this parameter is adjustable and offers a scale to prioritise the speed of compression.
- **Data type**: it refers to the characteristic of data, its entropy, redundancy etc. lity. These parameters significantly determine the amount of speed used during the compression procedure.

In summary the compression time in our situation will determine to evaluate the order of compression methods in relation to various set of files.

5.4.5 Average Compression Time (s)

This unit refers to the average amount of time taken to compress a set of data, measured in seconds. To calculate the average compression the following equation is used:

$$AverageCompressionTime(seconds) = \frac{TotalCompressionTime(seconds)}{Numberof DataPieces}$$

Retrieved from the equation it gives the sum of each individually compressed data divided by the total amount of data files. Accordingly if we have 10 data and the total compression time is 50 second, it will give 5 seconds per pieces in average.

5.4.6 Compression Speed (MB/s)

Compression speed in megabyte per seconds very much defines how fast each compression method can achieve the compression measured in the rate of the amount of megabytes transferred in each second during the compression. With other words, how quickly the compression algorithm can access the input data and produce compressed output. Compression speed can be determined with the following equation:

$$CompressionSpeed(MB/s) = \frac{Sizeofinputdata(MB)}{CompressionTime(s)}$$

This rate based on bytes per seconds that needed to be converted to MB by dividing the final output with 0.000001 according to this SI equation: 1 byte = 1000^{-2} MB.

5.4.7 Average Compression Speed (MB/s)

Using the compression speed equation above one can derive the average compression speed by measuring the total amount of compressed data in MB and divided by the total time taken to perform all compression process.

$$AverageCompressionSpeed(MB/s) = \frac{TotalSizeofCompressedData(MB)}{TotalCompressionTime(s)}$$

Average compression speed in practice can be important during file compression on a web server e.g. when an email with attached files needs to be compressed before transmission in order to consume less space. Speed is a crucial component in this scenario.

5.4.8 Decompression Time (s)

Decompression time is basically involves inverse mathematical computation that aims to extract the compressed file. It counts the elapsed time between the starting point, when the data is in compressed state, and the end point when the data turns back to its original state as it was before compression. When it comes to compression performance test, decompression does not weight the same manner. Buffering videos can be an example where decompression time is important, when the required data downloads from the server, it is necessary to decompress before use. Decompressing a file takes a reverse approach and, in comparison to compression it can only be done one way. Decompression time usually significantly shorter than compression time, due to several reasons:

- Algorithm complexity during compression involves more complex mathematical computation compared to decompression as it needs to analyse the data, find patterns, match them and apply various encoding techniques to reduce redundancy. Decompression algorithms construct the original data by using fewer computational steps. The way of reversing back a known process should consume less computational time.
- Allocating the available computation resources during compression to achieve highest possible compression ratio requires more computational resources, while decompression aims to priorities speed over efficiency.
- One place where decompression logic targets efficiency is optimisation. Compression algorithms apply more optimisation techniques for higher compression ratio.

5.4.9 Average Decompression Time (s)

Similarly to compression time, average decompression time takes all the amount of time it is necessary to decompress a file and divides it with the number of files. This metric is used to determine the performance of compression tool in terms of decompression

speed.

$$AverageDeompressionTime(MB/s) = \frac{TotalSizeofDecompressedData(MB)}{TotalDecompressionTime(s)}$$

The division of total decompressed data and total decompressed times gives the average decompression time. This can measure in case of e.g. five compression algorithm how much is the meidan time they use to extract the given sample data.

5.4.10 File Entropy

Entropy is a metric to determine what is the average content of information a file contains, rather how uncertain or random the data in a file. On the other hand it also defines the predictability of data; how regular the data occurs, what patterns can be identified. Therefore high entropy indicates high level of randomness, while low entropy signify the opposite. High entropy files are less compressible as they contain more random data. On contrary, low entropy files are more compressible due to their pattern redundancy. The goal of compression is to predict redundant patterns and reduce their occurrence, consequently if decompression successful it must closely resemble the original file. The estimation of file entropy in term of file compression and decompression typically need to follow some certain steps:

1. Calculating prior file entropy by using mathematical formulas before the compression. For this matter we have utilised histogram-based entropy (To et al., 2013).
2. Compress input file.
3. Measure compression ratio to evaluate the rate of the compressed size.
4. Extract the decompressed file.
5. Measure the entropy of the decompressed file.
6. Compare and evaluate the entropy values before and after compression.

If the value of decompressed data shows significant deviation compared with the original data that can indicate error in compression algorithm. If the original data is highly unpredictable and has a high entropy value e.g. encrypted files or random generated number sequences are difficult to compress (G. Y. Kim et al., 2022). To check the entropy of the dataset, one can utilise the following commands on the Linux command line:

```
sudo apt-get install ent
head -c 1M /dev/urandom > /tmp/out
ent /tmp/out
```

5.5 Data Visualisation

This section introduces various methods that are utilised after all the required statistical data are consolidated. Visualisation tools for statistical analysis serve to communicate research results, enhance understanding them and also convey complex information in a

clear manner. There four types of visualising tools are used in this project, these will be listed and briefly explained afterwards.

5.5.1 Statistical data tables

The main purpose for using tables to compare the performance of compression and decompression data from multiple aspects. The provided table summarises the **average** data sets in regards all tested file samples (10) and valid to column Compression size (MB), Compression Rate, Compression Time (s), Compression Speed (MB/s), Decompression Time (s), Decompression Rate, Decompression Speed (MB/s). The numeric values are derived after each compression method ran all the test rounds (9) and the repetitive rounds (15) in case of each file type. The following aspect are taken into consideration when the tables are constructed:

- **Compression efficiency evaluation**: The table provides exact information how effectively each file performed during execution and what are the differences between the compression and decompression from the aspect of time, rate and speed.
- **Compression algorithm comparison**: Examined the Compression Time (s), Compression Rate and Compression Speed (MB/s) and their average values, the performance of different compression algorithms are concluded. Accordingly, some categories e.g. fastest, slowest, most effective, most suitable etc. each compression method are assigned. Thereafter some use case recommendation are given.
- **Data characteristic evaluation**: Column Number of Characters and Entropy contains accurate data from the raw data. Knowing the structure of each uncompressed file helps to rank and match compression methods to specific file types.
- **Performance analysis**: Measure the average and overall average performance of each compression method on the overall set of files. It helps to understand the performance overhead and to optimise resource allocation on a later stage if necessary.

Occasionally, multiple smaller summary tables are used to draw graphs. These instances with the required data are directly harvested from the source data tables. A colour scheme is implemented and assigned between the slowest and fastest metric starting with red and finishing with green respectively. The medium values follow the colour orange that turns into red towards slowest value, and turns into yellow finally green as they reach the best or fastest value.

5.5.2 Graphs and charts

In order to represent data correlation between each files and the different compression methods graphs and charts are used, where it was suitable. These tools are tend to be constructed to help data visualisation and identify patterns and trends in a more intuitive way. Graphs allow to compare various data by defining in our case the following:

- **Compression efficiency**: Using line graphs or scatter plots to draw several chart and determine how efficient each compression tool on different file ranges by evaluating Original Size (MB) vs. Compressed Size vs. Compression Rate columns.
- **Elapsed time**: Using bar chart enables an easy comparison about the Compression Time (s) vs. Decompression Time (s) to highlight the measured differences achieved across different files.
- **Compression and decompression comparison**: Using multiple lines graph enables to compare the overall performance of each compression method and helps to visualising the overall performance by using different measurement data: visualised in the same graph.

5.5.3 Quadrants

One way to visualise how each compression algorithm performed is applying quadrants. This form of analysis allows the researcher to divide and present data in a structured manner. Quadrants can display data in a wide scale of four different dimension and allows the user to take advantage of utilising measurement pairs with their positive-negative attributes. The quadrants can be interpreted as follows:

- **Y-axis - Compression size (MB)**: Represents in what grade the compression tool reduced the size of the raw files.
- **X-axis - Compression time (s)**: Exact time elapsed while the algorithm produced the compressed file.

With that said, the ten observed data file sample in case of applying Compressed Size (MB) on the y-axis (efficiency), and Compression Time (s) on the y-axis (speed) by using the accurate position of each can finalise in four result: fast and efficient, fast but inefficient, slow and efficient, neither fast nor efficient. The scale of y-axis ranges 0-100 MB and represents what is the compressed file size in proportion to the original file size. A higher position on y-axis indicates better compression efficiency where most probably the compressed file size is significantly smaller than the raw file. X-axis represents the elapsed compression time is taken by the algorithm. Compression methods that are slow, positioned on the right half of the quadrant. The quadrant can be applied in different scenarios. In our first approach we examine each compression tool separately and place them in the quadrant to determine their characteristics. In the second approach we repeat the latter mentioned procedure on all five compression algorithms to summarise their overall performance.

5.5.4 Byte Value Histogram

Applying byte value histogram analysis gives good opportunity to understand the characteristics of files by capturing the most possible information about the processed data after compression and decompression. The primary goal is to analyse the distribution of bytes and therewith estimate the efficiency and redundancy of each algorithm in order to

find further optimisation possibilities, eventually determine the most suitable compression level for our goals. To more precisely present how the tested compression algorithms perform on byte level, a python script is implemented to measure the state of byte structure before and after compression. The methodology as follows:

1. Extract the byte values from the sample file before compression.
2. Extract the byte values after compression.
3. Count byte frequency by iterating through the compressed data and count the occurrence of each byte value.
4. Plot histogram by placing byte values on the x-axis and frequency counts on the y-axis.
5. Evaluate the results.

There are two main characteristics occurs along compressed and decompressed data analysis. If the byte value distribution is equal it indicates that the file is compressed or encrypted. Single peaks are mostly hints which byte sequences carries important information. A peak with 0x00 bytes can be ignored as it contains padded values.
In summary, byte value histogram analysis helps to identify patterns or anomalies in the compressed data. By interpreting the research result one can effectively categorise and recommend which compression method suit better on the given file type. Notice that histograms are only applied where they served purpose for the researcher.

5.6 Zip Statistical Analysis

5.6.1 Zip Compression Performance Analysis

We can safely state that the zip compression method is one of the most known tool for file compression and archiving. It is convenient to extract files with zip, most operating system includes by default this utility. After version integrity check the *man zip* command prompted us all the available operators that is to be of our interest to check more precisely their capability. Zip has a scale of 1-9 compression levels, where 1 is the fastest mode and 9 offers the highest compression rate. For this purpose all available options are tested by a BASH script (3.5.1) that is constructed to utilise nested loop to go through the raw data set. The inner loop takes the first data file and runs compression level 1-9 on it and records the compression time, compression rate and compression speed (bytes/s). Meanwhile the main loop repeats the whole code 15 times to collect statistically enough amount of data. Zip handles relatively easily the given file set, however its syntax needed to be adjusted with a *-q* parameter in addition to run the compression smoothly.

```
# Zip
start=$(date +%s.%N)
zip -"$level" -q "${filename_no_extension}_$level.zip" "$input_file"
end=$(date +%s.%N)
```

The most comprehensible data, Compressed File Size (MB) column shows a wide range

scaling from 12 MB to 69 MB with and average of 28.34 MB (Table 5.1). The most outstanding performance [6] indicates that *.json* files due to their organised structure, are easier to compress with good result then others. *Pcap* files [8, 10] however are more robust and zip only achieved to compress them approx. half a size in average with a compression rate between 1.5-2.4 (Figure 5.7). With that said, compressing a 100 MB *.pcap* file in the worst case [10] shows poor performance resulted to a 69.29 MB *.zip* file. This might be quite a large size for a single log file to transfer and does not indicate significant difference from the original. In spite of this, the compression time (3.16 s) of [10] seems to be fair and takes almost 50% less time than in case of a very similar file [8].

Table 5.1: Zip compression metrics

zip						
File nr.	File Name	Compressed size (MB)	Compression Time (s)	Decompression Time (s)	Compression Ratio	Entropy
1	accesslog.log	12.3156	1.8701	0.5826	8.6211	5.3820
2	apachelog.log	19.8368	3.0280	0.9468	5.3200	5.3955
3	connectionlog	22.1576	3.6858	0.7180	4.7478	4.7129
4	csvhoneypots	27.1746	4.4884	0.8562	3.8700	3.7589
5	fileslog.log	27.7080	2.9448	0.8864	3.7867	5.0336
6	honeypots.json	6.4262	1.4851	0.4822	16.4533	5.0182
7	httplog.log	17.7120	1.9716	0.6150	5.9322	5.5072
8	mawitraffic.pcap	44.7747	5.5921	0.8360	2.4225	6.9621
9	phishingemail.csv	36.0109	3.9398	1.0652	2.8889	5.3976
10	snortlog.pcap	69.2989	3.1667	0.9501	1.5100	7.7732

In some scenarios compression time is a crucial factor and zip performs better especially on some of the light-weight log files [1] an [7] than on the heavier ones [4] [8] [9]. In the case of [1] [6] [7] zip achieved an average compression time under 2 seconds. The overall compression time average (Figure 5.6) shows that all files performed quite well, under 6 seconds. The contrast between the compression time and decompression time is also significant. It needs to be noticed that speed performance itself does not reflect whether the file is compressed efficiently and it can only be interpreted in relation to compression ratio.

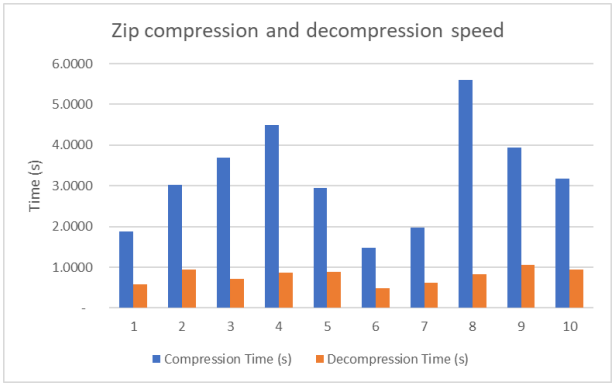

Figure 5.1: Zip compression performance

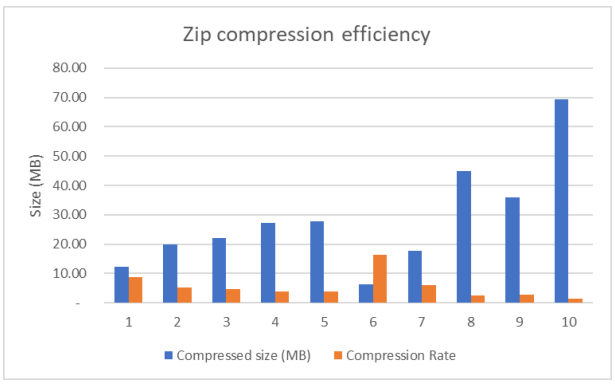

Figure 5.2: Zip compression efficiency

5.6.2 Zip Decompression Performance Analysis

During the *unzip* utility BASH script implementation some errors encountered due to the fact that it was not included in the Ubuntu Server environment. This problem is solved by manually installing it from the repository. The standard use of decompression syntax also throws syntax error and needed to be adjusted with a *-j* operator to move the unwanted command line output to junk folder.

```
# Unzip
start=$(date +%s.%N)
unzip -j "${filename_no_extension}_1.zip" -d "${filename_no_extension}_uncompressed"
end=$(date +%s.%N)
```

Unzip command offers an option to explicitly define where to store the decompressed data to undertake e.g. entropy analysis. For this purpose all raw data ran through an entropy analysis before and after the compression to detect any inconsistency. The results showed only 0.1 deviation on [10] that can be ignored. The files prior and after compression contained the same amount of data. The evaluated decompression time scales between 0.58 and 1.06 and indeed shows some deviation. File [1] is the fastest and [9] is the slowest, takes twice as much time to extract data. This result is not unexpected regarding the file structure complexity. Comparing these results with the compression time, the decompression process takes 5-8 time less. These time units can not be said significant.

5.6.3 Zip Quadrants Analysis

The overall performance (Figure 5.3) on all file sample, it clearly shows that zip offers a reliable solution in terms of compression time vs. compressed file size or compression rate. In our scenario 8 files positioned in the lower quadrant, that is more towards to a safe and optimised way of file compression where speed is important but not superior. There are two *.pcap* file [8] and [10] both a bit far away from their group. The compression size metrics of these two files indicates that *zip* utility process them within a reasonable time frame comparing to the larger part. However their compressed file size are outstandingly large in comparison to the average (5.6.1). Especially [10] which is however achieved on the 5th place in relation to compression speed, its size only shrunk with around 44.32% less in average.

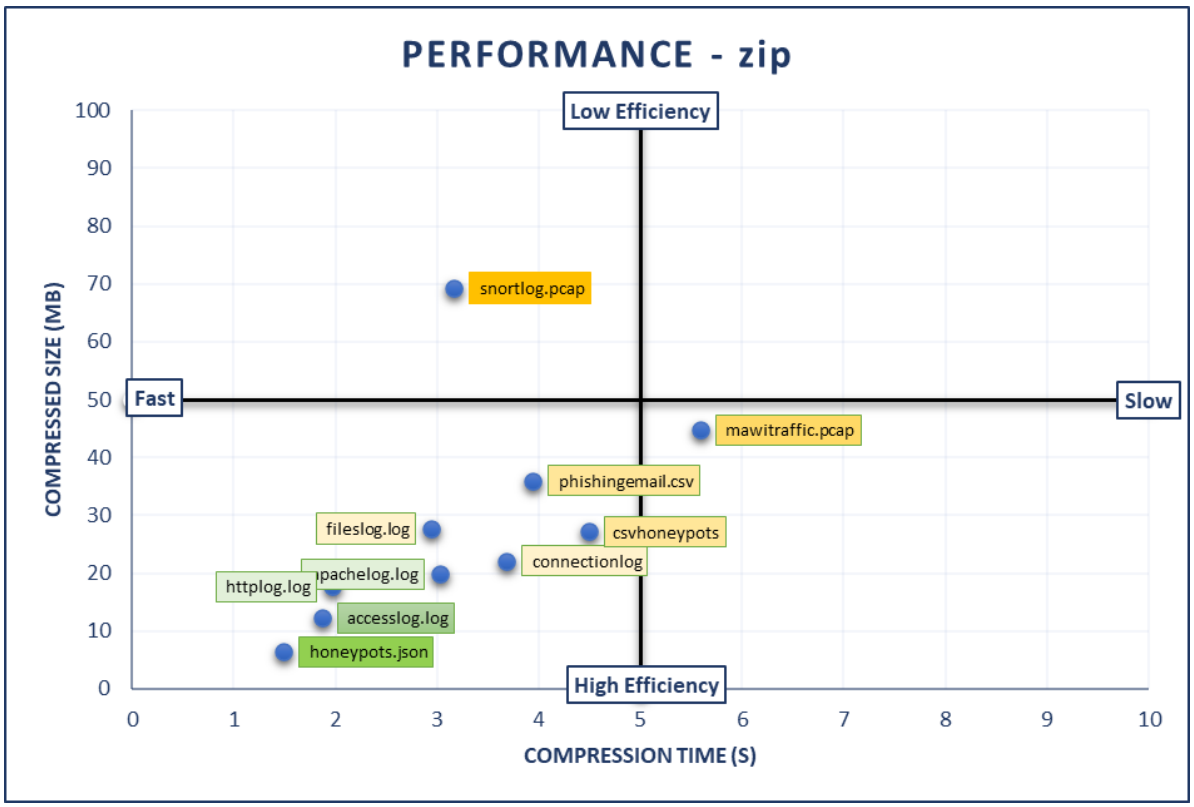

Figure 5.3: Zip performance quadrants

5.6.4 Zip Byte Value Histogram Analysis

Conducting further performance analysis on *phisingmail.csv* the compression performance by utilisation the default compression level (6) we can gain more in depth information about how the structure of the compressed file changed during compression. For this purpose a byte value histogram is taken first from the original file (Figure 5.4), thereafter from the compressed file (Figure 5.5). The purpose of this experiment to find out how *zip* achieve the highest compression ratio on this file.

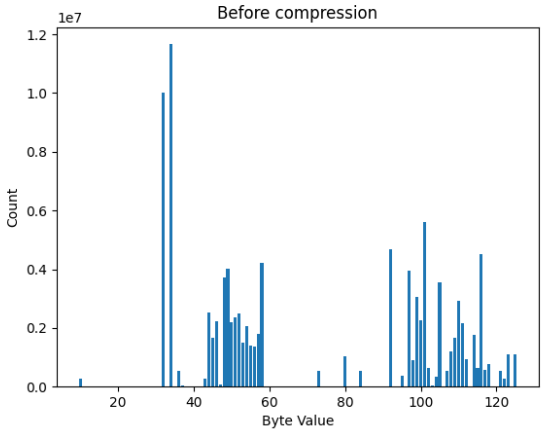

Figure 5.4: Phishingmail.csv before compression

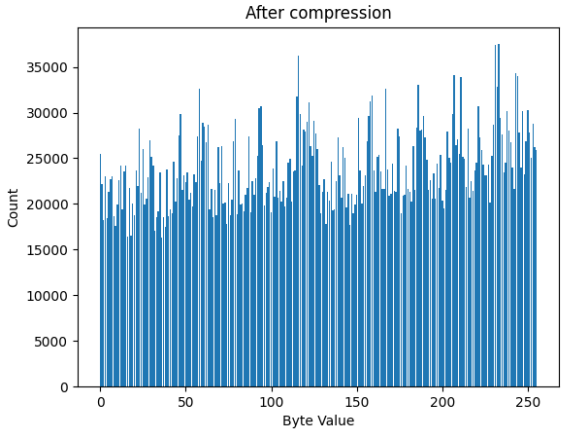

Figure 5.5: Phishingmail.csv after compression

On the first histogram we can observer a uniform byte distribution with high peaks and low values in a byte value range of 0-125 with a complete disordered structure. Some byte

values are missing some has extremely high frequency counts on Figure 5.4. Byte values data also show some inconsistency. Text files with large uniform areas can be compressed well with Huffman coding (part of the DEFLATE algorithm). When zip algorithm approach the data it assigns short code for long values and vica verse, thereby arranging the byte values into a more organised structure (Figure 5.5). As a result to this arrangement, most frequent values are made larger in order to establish a balanced state on the compressed file.

5.6.5 Summary

All in all the zip compression test produced good average results. Generally, it is hard to define which compression level is the most recommended as it depends on what the user need. The trade off here is the speed vs. compression rate as it is challenging to find both fast and efficient compression on the same time. The reason behind is the complexity of each level. While level 1-3 do not utilise high performance of mathematical algorithm, level 8-9 prioritise compression ratio over speed. *Zip* uses simpler algorithm like DEFLATE with fewer compression passes or less aggressive compression strategies when level 1-3 is chosen and spends less time to analyse the data. On the other hand, *zip* algorithm sacrifices more memory and CPU power to achieve higher compression rate. Therefore low compression levels are useful during file transfer or when low-powered devices at present. If the compression and decompression must be done with maximum efficiency, higher compression levels are recommended.

5.7 Bzip2 Statistical Analysis

5.7.1 Bzip2 Compression Performance Analysis

During version integrity check our system was lack of available distribution, so this tool had to be installed manually as well as zip. Ubuntu Server environment probably supports only minimal configuration by default. The BASH script is executed in normal manner and after the first round generated all the expected 122 files (90 compressed, 32 statistical). The compression syntax follows an easy logic, implementing the *-c* operator as well to redirect results of any given file, while keeping the original file.

```
# Bzip2
start=$(date +%s.%N)
bzip2 -"$level" -c "$input_file" > "${filename_no_extension}_$level.bz2"
end=$(date +%s.%N)
```

Bzip2 is ultimately constructed to achieve high compression rate mostly only on single files. It's compression level parameters scales from 1-9 where 1 is the fastest compression that rarely delivers good compression rate results, and level 9 that compresses best. As *bzip2* applies block compression mechanism, our expectation was that mainly compression rate performance is going to be dominated.

Starting with Table 5.2 the compressed file size overview that ranges between 4.01-68.67 MB, we can state that files with less complex structure [1] and [2] or higher hierarchical structured [6] are in favour of *bzip2* method. This makes certainly sense as block-compression tools match easier continuously repetitive patterns.

Table 5.2: Bzip2 compression metrics

bzip2						
File nr.	File Name	Compressed size (MB)	Compression Time (s)	Decompression Time (s)	Compression Ratio	Entropy
1	accesslog.log	7.4730	13.2978	2.4436	13.0533	5.3820
2	apachelog.log	10.8152	10.3111	2.8389	8.6578	5.3955
3	connectionlog	17.2036	11.0514	2.8507	5.9522	4.7129
4	csvhoneypots	19.0849	10.1193	3.4049	5.1767	3.7589
5	fileslog.log	24.6678	12.4373	3.4717	4.1722	5.0336
6	honeypots.json	4.0162	11.8599	1.8201	24.5044	5.0182
7	httplog.log	14.6503	13.3247	2.6058	6.9678	5.5072
8	mawitraffic.pcap	37.9705	8.9056	4.4033	2.7662	6.9621
9	phishingemail.csv	28.0198	9.2470	4.1343	3.4833	5.3976
10	snortlog.pcap	68.6713	11.6548	5.6219	1.4900	7.7732

Figure 5.7 show great performance on [1] and [6], average compression results on [3] [4] [7] and outstandingly poor on [10]. File [6] is compressed significantly with a compression ratio of 24.66. This is a very high rate value and comparing to the original size file the compression resulted in a 4 MB *.bzip2* file. Compression time metrics (Figure 5.6) do not show high peaks, *bzip2* mostly produced a good average without any outstanding performance on a scale between 8.9 seconds and 13.32 seconds. This range indicates a functionally stable compression tool, that prioritise compression rate over compression speed. This especially true for *.json* files which often contains repetitive keys or values that can be effectively compressed with block compression tools.

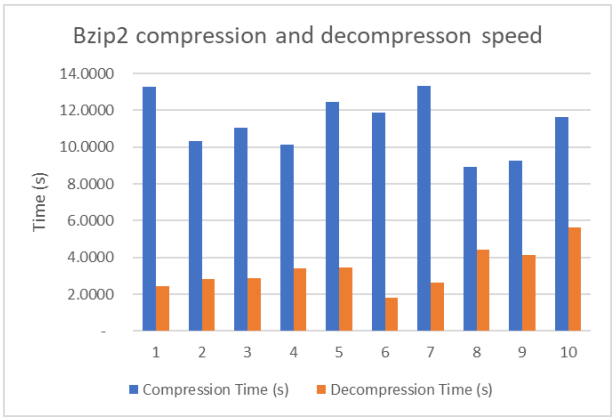

Figure 5.6: Bzip2 compression performance

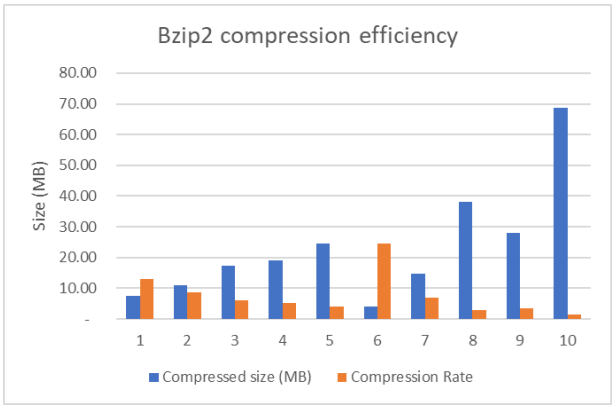

Figure 5.7: Bzip2 compression efficiency

5.7.2 Bzip2 Decompression Performance Analysis

For file extraction *bzip2* utility provides the *bunzip2* tool that solely focuses on decompression tasks. For BASH scripting we rather choose the built-in option which more suitable for compression and decompression purposes. Operator *-d* stands for decompress and *-k* keeps the original file after decompression stops.

```
# Bzip2 decompression
start=$(date +%s.%N)
bzip2 -d -k "${filename_no_extension}_1.bz2" > "${filename_no_extension}-uncompressed.txt"
end=$(date +%s.%N)
```

All finalised metric are also redirected to a separate file to ensure further implementation. The results after consolidation scaled between 1.8 and 5.6 seconds. File [6] placed first, followed by file [3,6]. File [10] placed in the last position and was three times slower than the first one. This difference can only be significant during large file decompression. Apart from this slight deviation the overall average is around 3.35 seconds. Decompression time in terms of security log file data does not carry too much weight at this size range. However in real-time systems such as streaming application or network communication fast decompression is crucial. Delays in e.g. large datasets decompression can cause latency in the overall response of the system. *Bzip2* seems not to be build for this purpose, rather a solid compression method for archiving files and optimise storage space.

5.7.3 Bzip2 Quadrants Analysis

Observing the overall statistical metrics (Figure 5.8) on the *bzip2* quadrants graph 5.8, the elements are spread in the middle of the lower two quadrants which clearly indicates performance over speed. This highlights the major characteristic of *bzip2* compression method. File [2] [3] [4] concentrated in a small circle around the lower part of the Y-axis containing similar statistical averages while file [1] and [7] a bit further to the right, also marked as fast and well-compressed.

On the left upper quadrant there is only one file [10] that falls into inefficient and slow category. As *.pcap* files contain sequence of packets with variable content and high diversity, most compression methods use additional time to compress them effectively. This is due to that algorithms rely on identifying and eliminating redundancy that is time consuming process.

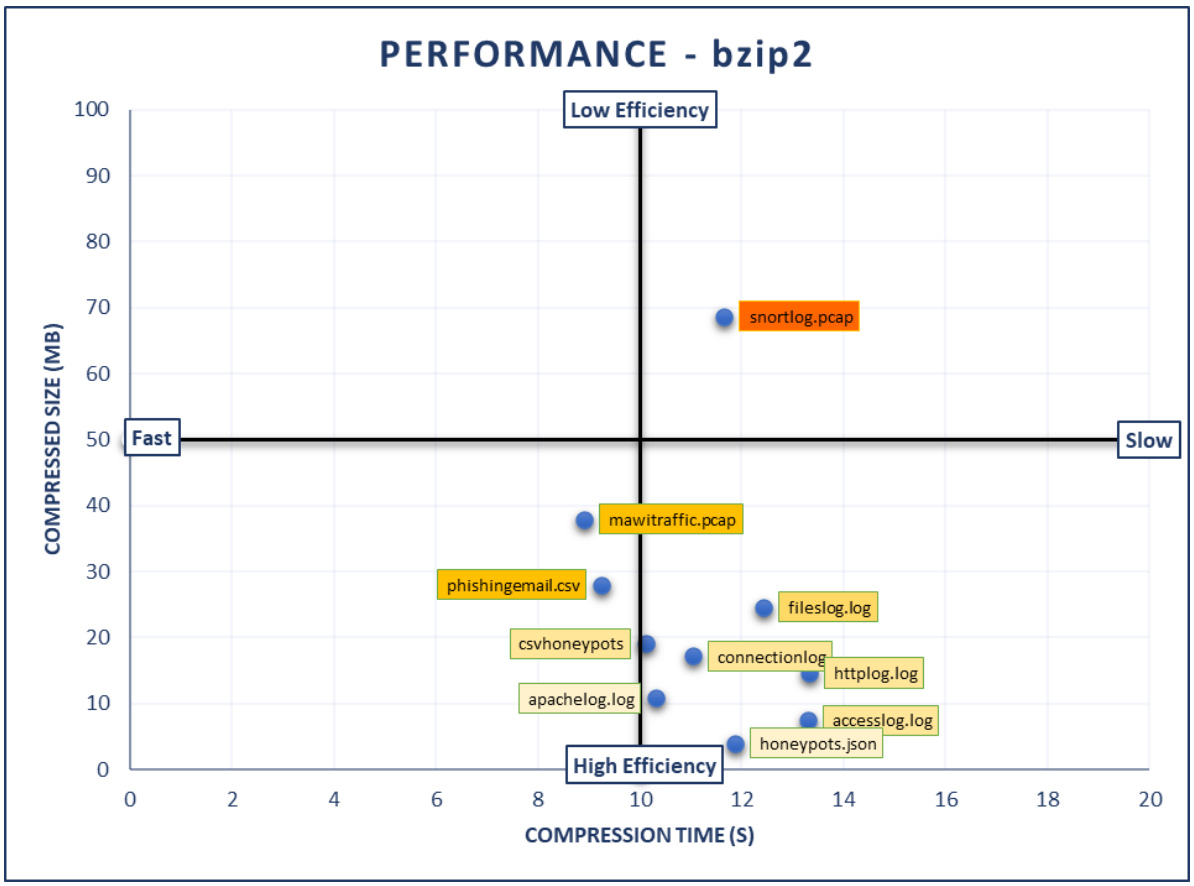

Figure 5.8: Bzip2 performance quadrants

5.7.4 Bzip2 Byte Value Histogram Analysis

The histogram of the *snortlog.pcap* file indicates some sort of order in comparison to the other bytes value histogram (5.6.4). On (Figure 5.9) we can see a relatively homogeneous spread of byte data frequency. This file is more likely contains equal amount of characters in organised lines in a well-structured pattern. There are a few exceptional peak mainly around byte position 100-125, indicating longer lines with more characters.

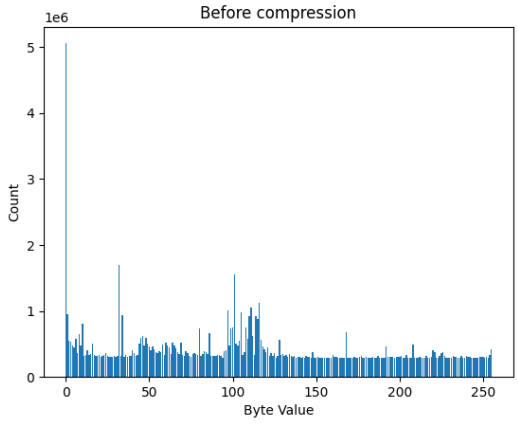

Figure 5.9: Snortlog.pcap before compression

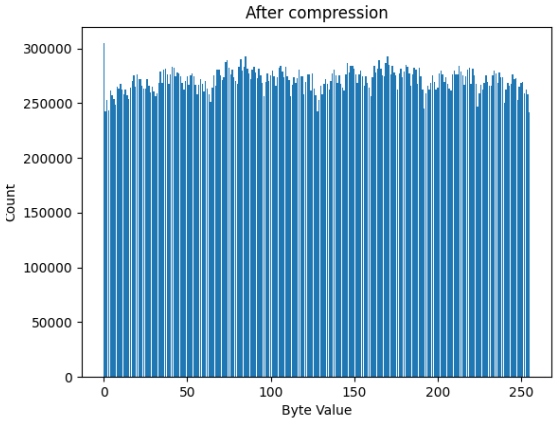

Figure 5.10: Snortlog.pcap after compression

The sample shows uniform distribution before compression, symmetrically forming two smaller bell-forms (Figure 5.9). After data compression (Figure 5.10) the frequency

has changed significantly, shrinking each byte segment and forming a more distributed pattern, with a way higher frequency count (250-300,000). Higher frequency means higher compression.

5.7.5 Summary

As *bzip2* open source compression algorithm utilise three different mathematical algorithm (Burrows-Wheeler transform, RLE, Huffman coding) for file compression, its is constructed to achieve high compression ratios, especially in case of text files. Therefore it more effectively reduces the size of each security data log file samples than *gzip* or *lz4*. Even thought, on some file types it could not manage to compress effectively. *Bzip2* indeed pays the price for achieving consistently good compression ratio. However its speed performance neither exceptionally fast nor exceptionally slow. It has a balance between compression efficiency and decompression speed. The effectiveness of block compression methods relays on several factors including mainly the structure and redundancy of the data. While file types like *.json* and regular log files benefit from block compression, *.pcap* files suffers in terms of efficacy. There are many reason like data entropy, level of repetition and block size that can interfere the compression process. The chosen compression level yet another component that plays important role to evaluate a compression algorithm correctly. According to the metrical result, *bzip2* is more ideal for storage utilisation, where maximum compression is a must. Definitely recommended for reducing the size for system logs, database dumps, all kinds of text-based data, especially the ones with consistent structure. Ideal for greatly reducing the size of backup files. When it comes to distribute data between networks, it is practical as long as we consider non-time sensitive scenarios. Developers often use this utility to compress source code repositories and to satisfy this needs *bzip2* is an excellent tool. Moreover, if storage space is limited or costly, this compression method offers good solution for challenges with long-term storage capabilities.

5.8 Gzip Statistical Analysis

5.8.1 Gzip Compression Performance Analysis

Gzip is a popular compression utility among the Open Source Community and most distros are shipped with it by default. The syntax works perfectly and by applying the *-c* operator it provides all the expected result in BASH. As *gzip* utility native on Linux systems the program interpreted without any error message. It's manual offers huge variety of compression options. The files are tested by the 1-9 compression level operators applying the same BASH script.

```
# Gzip
start=$(date +%s.%N)
gzip -"$level" -c "$input_file" > "${filename_no_extension}_$level.gz"
end=$(date +%s.%N)
```

The Compressed Size (MB) column shows (Table 5.3) a range from 6.42 to 69.29 MB that compresses a wide scale of results with an average of 28.34 MB. File [6] compressed very well, file [3] [4] [5] are in the same medium range position. Although the two *.pcap* files [8] [10] scores with a remarkable difference, placed at the end of the ranking. Interesting to notice that there is also a distinctive difference in between each *.pcap* files. File [8] was compressed 78.57% better than [10]. In line with this the overall compression rate shows some significant differences, highest is [6] with 16.45 and [10] with a very low metric (1.5) is the slowest.

Table 5.3: Gzip compression metrics

gzip						
File nr.	File Name	Compressed size (MB)	Compression Time (s)	Decompression Time (s)	Compression Ratio	Entropy
1	accesslog.log	12.3153	1.9546	0.5146	8.6222	5.3820
2	apachelog.log	19.8366	2.5351	0.6958	5.3200	5.3955
3	connectionlog	22.1574	3.4831	0.6193	4.7478	4.7129
4	csvhoneypots	27.1744	4.2795	1.1180	3.8700	3.7589
5	fileslog.log	27.7078	2.9345	1.0712	3.7867	5.0336
6	honeypots.json	6.4260	1.3202	0.7701	16.4533	5.0182
7	httplog.log	17.7118	2.1560	0.9952	5.9322	5.5072
8	mawitraffic.pcap	44.7745	5.9329	0.8863	2.4225	6.9621
9	phishingemail.csv	36.0106	4.1586	0.9564	2.8889	5.3976
10	snortlog.pcap	69.2986	3.5306	0.7511	1.5100	7.7732

One of the reason behind this deviation lies in the different file structure. Compression Time (s) column serves us with very attractive numbers. In average all compression tool achieved to shrink the sample files under 6 seconds (5.11). Regarding that various file types are used hence various operators are tested on the same file, this result can be considered good. There is a notable difference in speed vs. compression rate in between file [3] [4] and [10] [9].

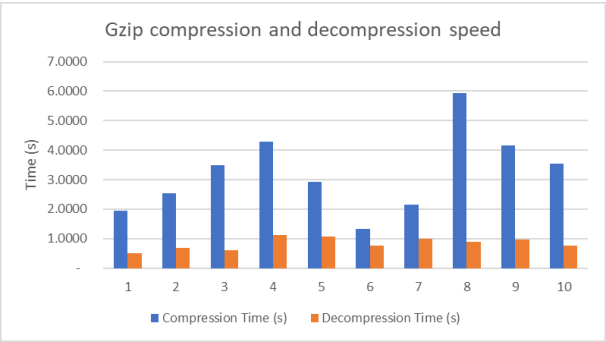

Figure 5.11: Gzip compression performance

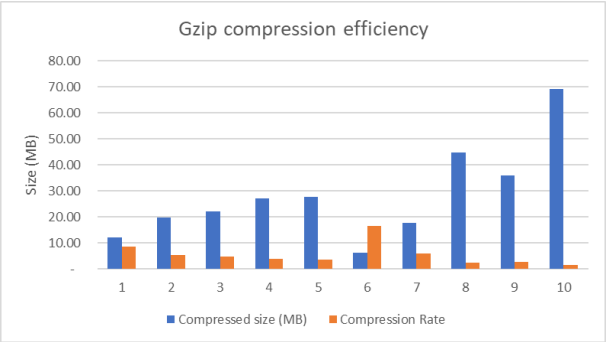

Figure 5.12: Gzip compression efficiency

The compression cycles are completed with very similar results (0.05 and 0.12 difference) in both cases respectively, while the compression rate shows significant difference in favour of file [3] and file [4]. *Gzip* managed to compress robust files [9] [10] and light-weight *.log* files [3] [4] with the same average speed, but with lower compression rate in average. On Figure 5.12 we can observe a normalised speed utilisation, except [8] which compression speed has taken three times longer than what [1] consumed despite that the latter has

lower entropy.

5.8.2 Gzip Decompression Performance Analysis

The decompression syntax for *gzip* worked out of box without any issue. The *-d* operator stands for decompression and the *-k* for keeping the original file.

```
# Gunzip
start=$(date +%s.%N)
gzip -d -k "${filename_no_extension}"
end=$(date +%s.%N)
```

Decompression Time (s) column represents a range of 0.51-1.11 seconds that seems to be a good overall result and it is in balance corresponding to compression speed. Interesting to notice that [1] [3] scored on the top, but [10] is faster then [6] which is a rare result even in relation to decompression speed. The entropy results shows no significant differences between them so far.

5.8.3 Gzip Quadrants Analysis

Gzip complies well in regards both compression and decompression speed. Having a look on the overall metrics (Figure 5.13) the proportion of files can be detected on the lower left quadrant.

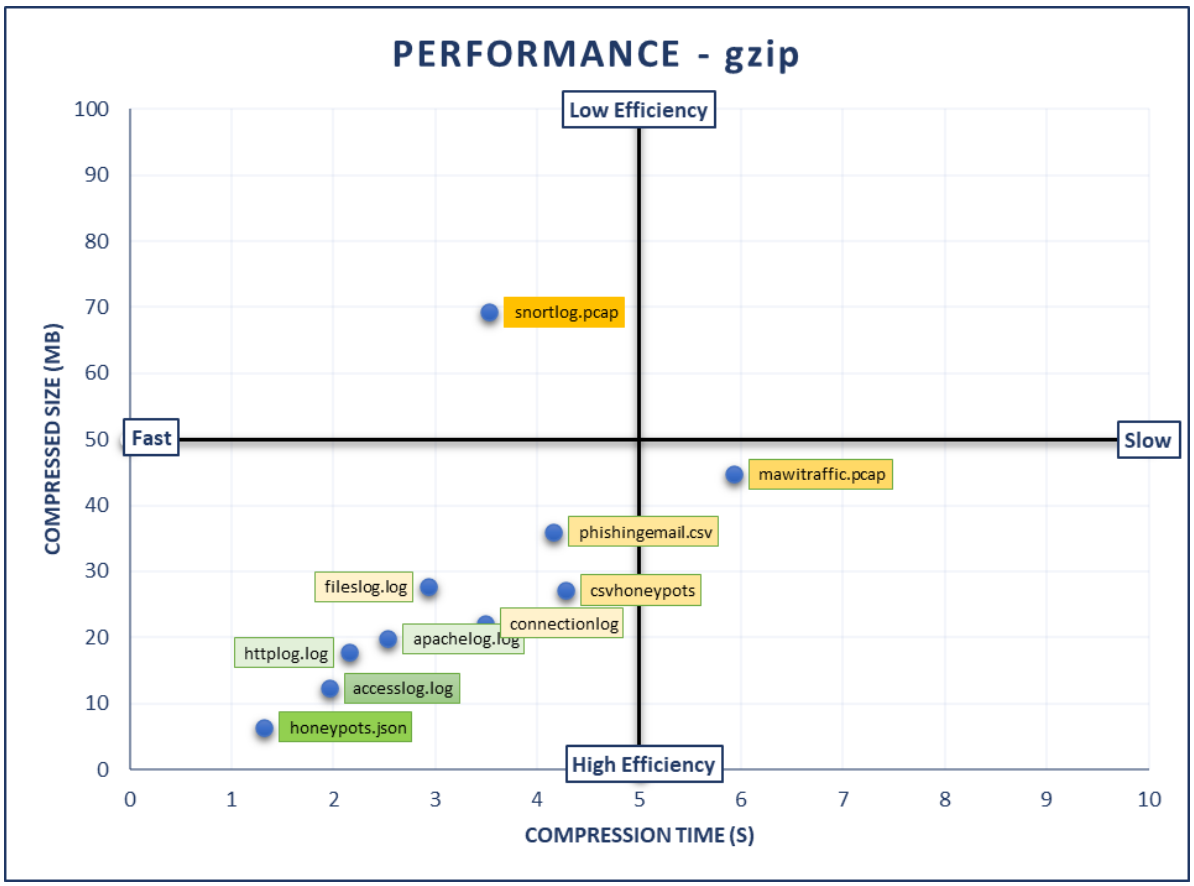

Figure 5.13: Gzip performance quadrants

It indicates that files [1] [2] [5] [6] [7] aligned on the same increasing trend line are falling into both high efficiency and low compressed file size category. This especially true for files [6] and [1], where the rate of speed vs. compression efficiency is the highest. There are only two files positioned further away from the other eight. File [8] achieved low rate and slow compression performance and [10] positioned on the upper left quadrant indicating slow and inefficient compression result. The overall performance signifies fast and productive compression rate in case of 80% of the total file sample.

5.8.4 Summary

Gzip is a single file compression utility and primarily designed for Unix-based systems and serves as a good all-around compression and decompression tool. The major mathematical algorithm used by *gzip* is DEFLATE, same as in case of *zip*. Therefore some of the testing results shows similarities in terms of compression time and compression rate. They likely behave the same way when it comes to compress different types of log files and process *.pcap* data. *Gzip* delivers a reliable good compression standard in terms of almost all files, except [10], but this file due to its structure caused difficulties to all the tested compression algorithms. Generally this compression tool offers relatively high compression ratio within a reasonable time frame for all types of files to archive. It preserves the file attributes during compression and extraction and easy to handle, especially in Linux environment. There are unfortunately some challenges by using *gzip*. As a standalone compression tool can not archive directories, in such case, it must be combined with the *tar* archiving utility. This makes its usability a bit more complicated for beginners. Furthermore it might not be enough fast for real-time data compression, here *lz4* perform much better. The overall scores indicates that *gzip* can achieve reasonable compression vs. speed performance on 80-90% of the tested security log types.

5.9 Lz4 Statistical Analysis

5.9.1 Lz4 Compression Performance Analysis

The *lz4* compression algorithm belongs to the LZ77 family and it is known from its fast performance. Along testing the program executed in a way more faster pace then the other. Significant error is not encountered. The syntax allows to create each compressed file on the disk for further statistical analysis. The program logic similar to what is described in section 5.8.1.

```
# Lz4
start=$(date +%s.%N)
lz4 -"$level" -c "$input_file" > "${filename_no_extension}_$level.lz4"
end=$(date +%s.%N)
```

Table 5.4 summarises the data gathered during the test. The findings show a narrow range 0.6-2.4 seconds in terms of compression time, where basically all file are successfully

compressed under 2.5 seconds. File [6] way ahead in speed from the other nine with 0.66 second time. The overall average is 1.57 seconds, surprisingly good result at this point and seven files scored under 2 seconds compression time (Figure 5.15). When it comes to measure the compressed file sizes, *lz4* does not show exceptionally outstanding metrics.

Table 5.4: Lz4 compression metrics

lz4						
File nr	File Name	Compressed size (MB)	Compression Time (s)	Decompression Time (s)	Compression Ratio	Entropy
1	accesslog.log	14.8252	1.1178	0.1349	7.1689	5.3820
2	apachelog.log	22.0881	1.4565	0.1319	4.8133	5.3955
3	connectionlog	28.2418	1.6090	0.1682	3.7256	4.7129
4	csvhoneypots	33.9520	1.9443	0.1793	3.1644	3.7589
5	fileslog.log	35.7941	1.3274	0.2171	2.9356	5.0336
6	honeypots.json	8.0469	0.6650	0.4257	13.1444	5.0182
7	httplog.log	22.3005	1.0741	0.1308	4.7167	5.5072
8	mawitraffic.pcap	48.2289	2.0972	0.1732	2.2593	6.9621
9	phishingemail.csv	42.3556	2.4013	0.4511	2.4800	5.3976
10	snortlog.pcap	68.7497	2.1033	0.1541	1.5167	7.7732

Figure 5.14 indicates, that the algorithm is struggling to deliver good compression ratio in accordance to [8] [9] [10]. Moreover we can not state that the result for [4] and [5] is satisfying either with 33.95 and 35.79 seconds respectively. Surely the result for [6] represents an outstanding compression ratio (13.14), but the remaining files have a ratio under 5.

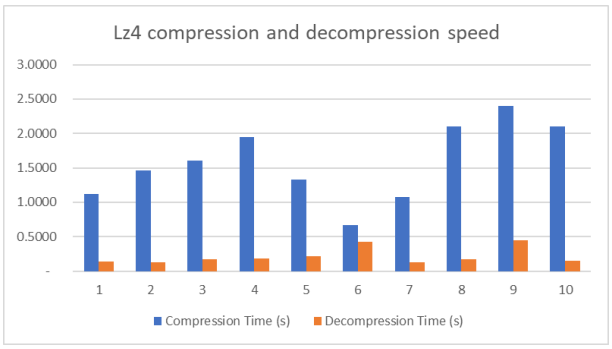

Figure 5.14: Lz4 compression performance

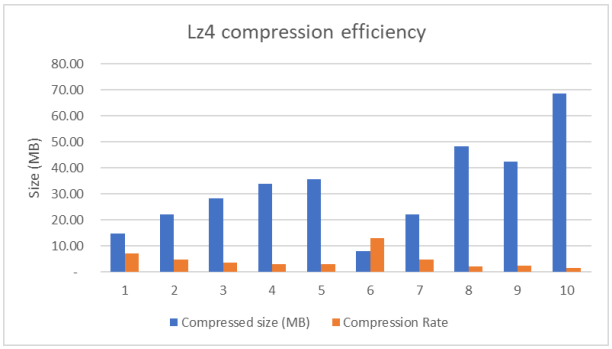

Figure 5.15: Lz4 compression efficiency

As it is discussed above, *lz4* is rather determined to get the work done fast than to offer reliable file compression rate. The entropy comparison shows no inconsistency the before and after results are matches exactly.

5.9.2 Lz4 Decompression Performance Analysis

The BASH decompression snippet basically follows the same pattern as in case of the other two open source utility.

```
# Lz4 decompression
start=$(date +%s.%N)
lz4 -d -k "${filename_no_extension}_1.lz4" "${filename_no_extension}_uncompressed"
end=$(date +%s.%N)
```

Seemingly the Decompression Time (s) column demonstrates very low metrics scaling 0.13-0.45 seconds. It implicates noticeable differences between the compression and decompression time. In the best case [2] it takes ten times shorter (0.1319s) to extract the same file. However some numbers especially [6] [9] give a bit controversial outcome from what we are expected. File [6] which performance was remarkably outstanding in the previous tests, outperforms in comparison with others. Yet in most cases file extraction does not have the same importance as compression. Light-weight files like [1] [2] shows convincing results.

5.9.3 Lz4 Quadrants Analysis

It might have been enough to use a single quadrant to represent the overall scores in accordance to *lz4* test findings. Nine files are positioned in the lower left quadrant as all of them, especially [1] and [6] achieved high speed and high efficiency.

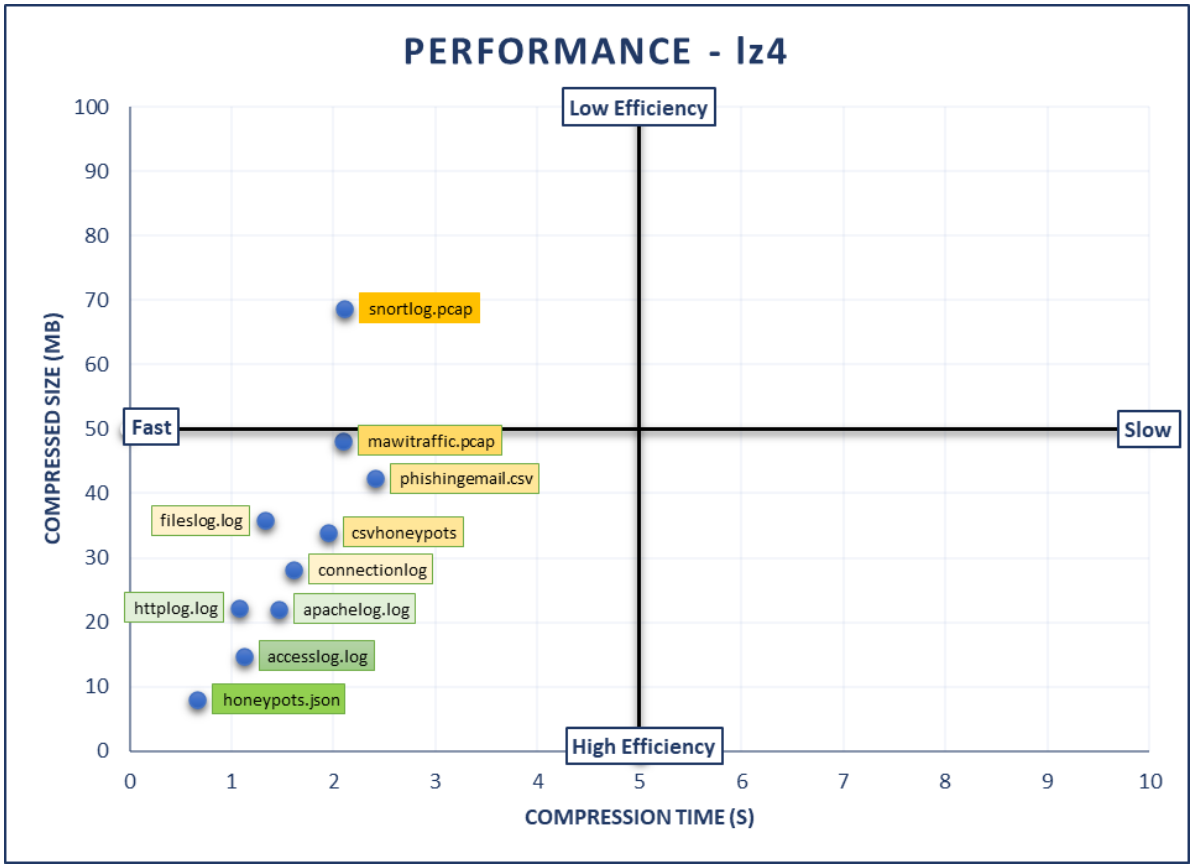

Figure 5.16: Lz4 performance quadrants

The two *.pcap* gave more challenges for the algorithm. File [8] is compressed to 48.22, with that said it has been shrank half the size. File [10] with a size of 68.74 MB and 1.51 compression ratio. These two results are reflecting that *lz4* is most suitable for light-weight file compression, and it does its duty in a quite fast manner. If we take a closer look at the range of 0-2.5 on x-axis, [6] [1] [3] [7] [9] aligned in an almost straight increasing line with proportional difference in between each other. This can indicate that simple security log files such as *.json* and *.csv* that contain structured repetitive data are tend to be compressed faster. The reason behind is that *lz4* utilises relatively simple mathematical algorithm that has low computational overhead and lack of text modelling. More importantly it does not perform backtracking and by that saves time on symbol parsing. This also explains why more complex structured files [8] [10] with lower entropy perform worst in terms of compression rate, regardless the chosen operator level. This summarised graph (Figure 5.16) outlines well the main characteristics of *lz4* which is speed over compression ratio.

5.9.4 Summary

When speed plays an important role during e.g. file transfer, real-time streaming or in network protocols, the implementation of *lz4* compression method can be handy. It can deliver speed and accuracy on the same time, both compression time and decompression time results are outstandingly good. Compression rates show that definitely the lower numbers are in majority. What is especially challenging for this tool is to approach data with complicated structure and high entropy. This compression utility is not ideal for that, it rather should be used ins scenarios where not enough CPU power is available or the goal is not to maximise compression ratio, but do the work done fast. When security log data needs to be transmitted between different systems or should be hived for in a centralised repository to be analysed, the speed of *lz4* comes to play an important role. With this one can save time on log file collection, reduce file transfer overhead during conducting real-time incident response. In some other cases like Security Event Management (SIEM), *lz4* helps to respond more effectively by accelerating the process. These metrics highlight that the main application fields of this algorithm where high throughput and low-memory data processing is required.

5.10 7z Statistical Analysis

5.10.1 7z Compression Performance Analysis

The *7z* compression utility designed to work a bit different than the other open source tools, however it certainly follows the same logic with the level operators just like *bzip2* or *gzip*. The additional difference is the implementation of *-mx* in the script below. It specifies a variable for compression level incrementing from 1-9 and creating compressed file for each compression level. The program flow of *7z* worked differently and threw a lot of unnecessary information on the screen by default made it impossible to run the

script as planned. Therefore a small adjustment applied to redirect the stdout to */dev/null* directory.

```
# 7z
start=$(date +%s.%N)
7z a -mx="$level" "${filename_no_extension}_${level}.7z" "$input_file" > /dev/null 2>&1
end=$(date +%s.%N)
```

Certain compression algorithms build to achieve specific goals either high speed or high compression ratio, rarely both. *7z* is not an exception out of this. To find out how it performs we need to observe the metric results in Table 5.5.

Table 5.5: Lz4 compression metrics

7z						
File nr	File Name	Compressed size (MB)	Compression Time (s)	Decompression Time (s)	Compression Ratio	Entropy
1	accesslog.log	9.0345	19.0823	0.8318	11.7200	5.3820
2	apachelog.log	11.5566	23.6042	1.1548	9.3522	5.3955
3	connectionlog	19.5710	22.5750	1.4469	5.3844	4.7129
4	csvhoneypots	19.3793	39.6818	1.2979	5.5133	3.7589
5	fileslog.log	24.5637	24.4135	1.4767	4.2922	5.0336
6	honeypots.json	4.9470	10.1430	0.4120	21.2478	5.0182
7	httplog.log	15.4530	13.9217	1.0260	6.8089	5.5072
8	mawitraffic.pcap	30.5714	24.0378	2.1062	3.5706	6.9621
9	phishingemail.csv	22.2767	25.6600	1.7512	5.0244	5.3976
10	snortlog.pcap	48.3782	10.6421	2.9378	2.3611	7.7732

Usually the most valuable information for ordinary user is to how small the compressed file size is in comparison to the original file. There are some interesting test result, starting with the two best [1] and [6] where *7z* produced outstandingly good result 4.94 MB and 9.03 MB respectively (Figure 5.19). Notice that the *.json* file is compressed half the size of the *.log* file. The rest of the files are at a size of between 11 MB 50 MB. The heaviest file [10] is not compressed so bad either, *7z* utility could manage to achieve lower file size or higher compression ratio. Apparently *.pcap* file structures are demanding for this tool.

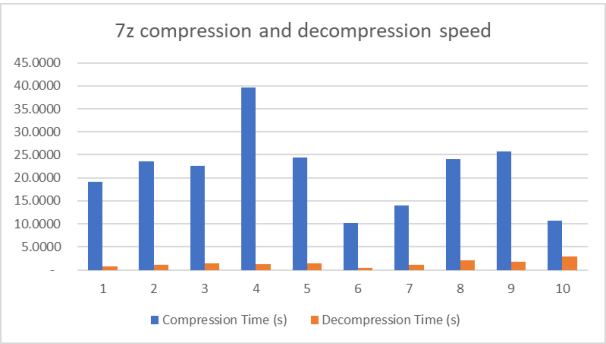

Figure 5.17: 7z compression performance

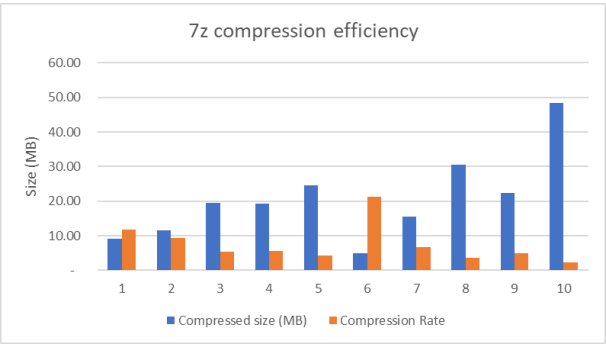

Figure 5.18: 7z compression efficiency

As we take a look at the Compression Time (s) column, we can experience larger numbers in all cases. The best [6] with 10.14 seconds, is just slightly faster than [10]. This indicates a bit contradictory result in accordance to the previously mentioned data measurement. One might suppose that in order to finalise high compression the compression mechanism

needs to sacrifice additional computational time. Logically, the higher compression rate we want, the longer it takes. Another surprising result represents a very slow speed performance on thee *.csv* file [4] (Figure 5.17), that is on the other hand attained a relatively well compression ratio (4^{th} position). It has been noticed that the overall execution time of *7z* algorithm in BASH script was disturbingly slow. It took over 5 hours to finish the script, however these measurement records are out of scope of this research. Entropy test does not indicate any abnormality between the in-and output files.

5.10.2 7z Decompression Performance Analysis

The *7z* command line utility offers multiple options for decompressing archives and compressed files. The *man* command comes useful in such situation. A bit of adjustment is taken on this snippet as well to eliminate the unwanted stdout data.

```
# 7z Decompress
start=$(date +%s.%N)
7z e "${filename_no_extension}_1.7z" > /dev/null
end=$(date +%s.%N)
```

The measurement metric shows the three fastest [6] [1] [2] whereas only the first two scored under 1 second (Figure 5.17). The slowest decompression time recorded in case of [10] which again contradicts with its exceptionally fast compression time.

5.10.3 7z Quadrants Analysis

The overall performance of *7z* algorithm on the security log data set indicates a slow but steady compression formula that does not deliver exceptionally high speed results, rather high compression ratio. It's efficacy relays on its quite complicated and robust algorithmic structure. Examining the metric data (5.19), six files are positioned on the right lower quadrant where high efficiency and small compressed file size are coupled with slow compression time. The *.json* file [6] seems to be the most compressible with a rate of 21.24 and after a noticeable gap in between followed by a *.log* file [1] with its 11.72 compression ratio. The second fastest [10] scored with the lowest compression rate. The files are spread in a bit larger radius, [5] [8] [9] are around the same spot which is a medium high compression time and a compressed size percentage of 25.8 MB in average. This is equivalent to shrink the original data with approx. 75%. This is not an outstandingly good result. Furthermore it is hard to draw a performance trend line here.

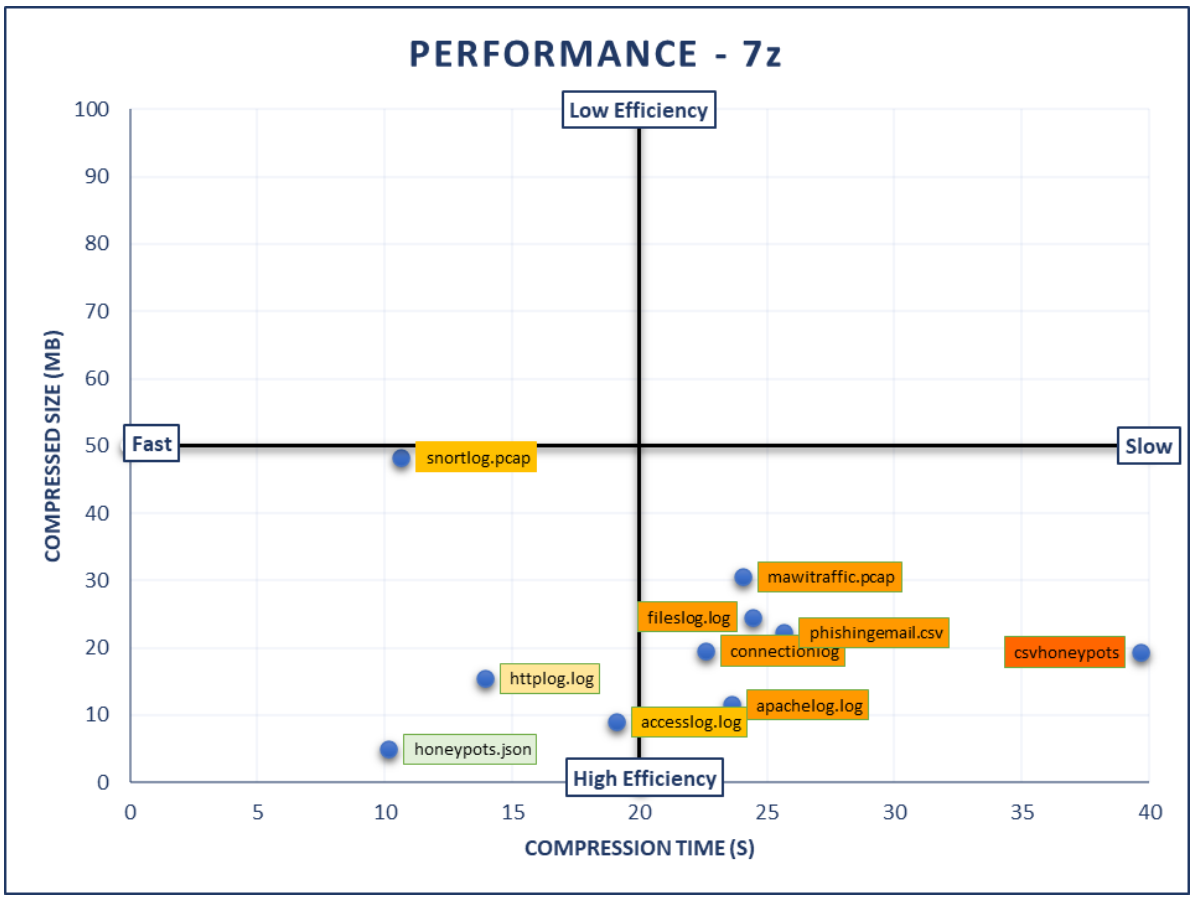

Figure 5.19: 7z performance quadrants

5.10.4 Summary

The LZMA (Lempel-Ziv-Markov chain algorithm) algorithm is build on high-compression promises and indeed capable to deliver this result, unfortunately at a price of unpleasantly slow execution. The simple reason behind is that *7z* applies dictionary-based algorithm that utilises sophisticated encoding processes, such as range encoding and binary trees, which demand additional computational resources. The utility consumes massive CPU power and struggles to compress complex structured files [4] [5] [8] [9] within a reasonable time frame. Even files with low entropy values [3] [4] [6] falls under this category. The consolidated compression rates indicate good result in relation to [1] [6], but the rest of the files are mostly scores on an average scale. Undoubtedly, *7z* delivers solid compression rates, however it is not clear where is the balance between the invested time and the expected outcome. There are certainly several positive technical aspect of implementing *7z* such as adaptive encoding which enables to dynamically adjust its compression parameters to the input data. This opens a wide range of dataset to work with. It also allows to take advantage of multi-threading function and use multiple processors simultaneously. This function can enhance its performance that can be a reasonable solution to save time on processing multiple security log dataset on the same time. *7z* is widely used for file backup, archival storage and software distribution and supported by various third-party software.

5.11 Overall Performance Evaluation

As all the compression statistical tests are done, the overall result are represented in this section. For better visualisation conditional formatting is implemented on the table structure to highlight the importance of each findings. The red colour indicates values that are below the threshold and not productive in relation to the assign research criteria. Orange is used where the values are approaching better productivity, but they are still closer to lower performance. The yellow colour represents medium or higher performance and can be considered as a good and acceptable average quality. Finally, green is equal to high quality performance, way over the so called average and very much recommended.

5.11.1 Overall Speed Performance

Table 5.6 summarises each compression algorithm and their performance in relation to compression and decompression speed. *7z* shows a steady but very slow accomplishment on all files. It performs fast compression on files [6] and [10] in comparison to the others. However these results can not be considered outstanding as all five algorithm achieved good speed test result on it. Its column, marked with red also represents that the overall scores just good enough to take the last place. The decompression speed on contrary represents medium results.

Table 5.6: Overall performance speed

File Info		Compression Time (s)					Decompression Time (s)				
Nr.	File Name	7z	bzip2	gzip	lz4	zip	7z	bzip2	gzip	lz4	zip
1	accesslog.log	19.0823	13.2978	1.9546	1.1178	1.8701	0.8318	2.4436	0.5146	0.1349	0.5826
2	apachelog.log	23.6042	10.3111	2.5351	1.4565	3.0280	1.1548	2.8389	0.6958	0.1319	0.9468
3	connectionlog	22.5750	11.0514	3.4831	1.6090	3.6858	1.4469	2.8507	0.6193	0.1682	0.7180
4	csvhoneypots	39.6818	10.1193	4.2795	1.9443	4.4884	1.2979	3.4049	1.1180	0.1793	0.8562
5	fileslog.log	24.4135	12.4373	2.9345	1.3274	2.9448	1.4767	3.4717	1.0712	0.2171	0.8864
6	honeypots.json	10.1430	11.8599	1.3202	0.6650	1.4851	0.4120	1.8201	0.7701	0.4257	0.4822
7	httplog.log	13.9217	13.3247	2.1560	1.0741	1.9716	1.0260	2.6058	0.9952	0.1308	0.6150
8	mawitraffic.pcap	24.0378	8.9056	5.9329	2.0972	5.5921	2.1062	4.4033	0.8863	0.1732	0.8360
9	phishingemail.csv	25.6600	9.2470	4.1586	2.4013	3.9398	1.7512	4.1343	0.9564	0.4511	1.0652
10	snortlog.pcap	10.6421	11.6548	3.5306	2.1033	3.1667	2.9378	5.6219	0.7511	0.1541	0.9501

Bzip2 that is a bit faster than *7z* in regards to all test metrics. Even though it performs worst on the above mentioned files, its overall numbers indicate that it is indeed at least twice as fast than *bzip2*. *Bzip2* is in a colour range where it may or may not considered to be implemented. Decompression speed is under our expectation, poor performance in comparison to its competitors. *Gzip* is above the average range and performs fast compression especially on [2] [5] [6]. It shows a good overall performance in both compression and decompression. *Lz4* compression algorithm is simple fast on all the tested files with significantly better results than the other four algorithm. Speed is the strongest side of *lz4* and therefore its deployment is highly recommended. *Zip* produced similar speed test result as *gzip* and can be considered as a good choice.

5.11.2 Overall Compression Efficiency

(Table 5.7) represent the core part of this research document as considerably the most important characteristics of compression algorithm is efficiency. *7z* show significantly better result on the data corpus, than the other tools. High performance in compression ratio and consequently reasonably smaller file size highlight its high efficiency on almost all the files, even on the complex *.pcap* file [10].

Table 5.7: Overall performance efficiency

File Info		Compressed Size (MB)					Compressed Rate				
File Nr	File Name	7z	bzip2	gzip	lz4	zip	7z	bzip2	gzip	lz4	zip
1	accesslog.log	9.03	7.47	12.32	14.83	12.32	11.7200	13.0533	8.6222	7.1689	8.6211
2	apachelog.log	11.56	10.82	19.84	22.09	19.84	9.3522	8.6578	5.3200	4.8133	5.3200
3	connectionlog	19.57	17.20	22.16	28.24	22.16	5.3844	5.9522	4.7478	3.7256	4.7478
4	csvhoneypots	19.38	19.08	27.17	33.95	27.17	5.5133	5.1767	3.8700	3.1644	3.8700
5	fileslog.log	24.56	24.67	27.71	35.79	27.71	4.2922	4.1722	3.7867	2.9356	3.7867
6	honeypots.json	4.95	4.02	6.43	8.05	6.43	21.2478	24.5044	16.4533	13.1444	16.4533
7	httplog.log	15.45	14.65	17.71	22.30	17.71	6.8089	6.9678	5.9322	4.7167	5.9322
8	mawitraffic.pcap	30.57	37.97	44.77	48.23	44.77	3.5706	2.7662	2.4225	2.2593	2.4225
9	phishingemail.csv	22.28	28.02	36.01	42.36	36.01	5.0244	3.4833	2.8889	2.4800	2.8889
10	snortlog.pcap	48.38	68.67	69.30	68.75	69.30	2.3611	1.4900	1.5100	1.5167	1.5100

Bzip performs well, above the average expectation and its metrics result coupled with its speed makes its position a bit stronger. *Gzip* also produced above the average, yet still has to improve especially when it comes to compress files [8] [9] [10] with high complexity. The overall performance is sufficient. *Lz4* comes with less productive compression rate results, rather converging to a minimum standard in terms of compressed file size as well. It is clear that not efficient enough to be considered in scenarios where high compression efficiency is a must. Finally, *zip* offers a reliable middle-way solution for daily file archiving. Nothing outstanding, but a good medium overall score.

5.11.3 Quadrants Evaluation

For the overall evaluation (Figure 5.20) a summative quadrant is created, where our goal is to visualise how each compression algorithm deviates from each other. The calculation of this quadrant presentation based on the previous statistical analysis. Each result is taken from the respective table column and implemented in a common quadrant graph allowing the reader to understand how their performance differs. By measuring the performance one can make recommendation about the application field of different compression methods.

Accordingly, the two most significant measurement unit are chosen that enables to describe the nature of a compression algorithm. The first is the Compression Time (s) assigned to x-axis and the Compression Size (MB) assigned to y-axis. This way we can set four evaluate criteria: fast and high efficiency, fast and low efficiency, slow and low efficiency and lastly slow and high efficiency. Accordingly the following results are determined staring from left to right:

- **Lz4**: Shaping a group primarily in the left lower quadrant, indicating fast and high

efficiency on easy structured files, towards less efficiency on files with higher entropy.

- **Zip**: Shaping a form also on the left corner of the lower left quadrant, presenting slower compression speed than *Lz4*, but better results in compressed size.
- **Gzip**: Most of the files are located in the lower left quadrant with lower speed result, but with almost the same compression efficiency as the *zip* compression method.
- **Bzip2**: Forming its group in the middle of the left lower quadrant, indicating slower compression speed than *lz4, zip* and *gzip*, but better compressed size results than the previous three.
- **7z**: Files are spread between the lower two quadrants resulting slow speed performance, but high compression efficiency.

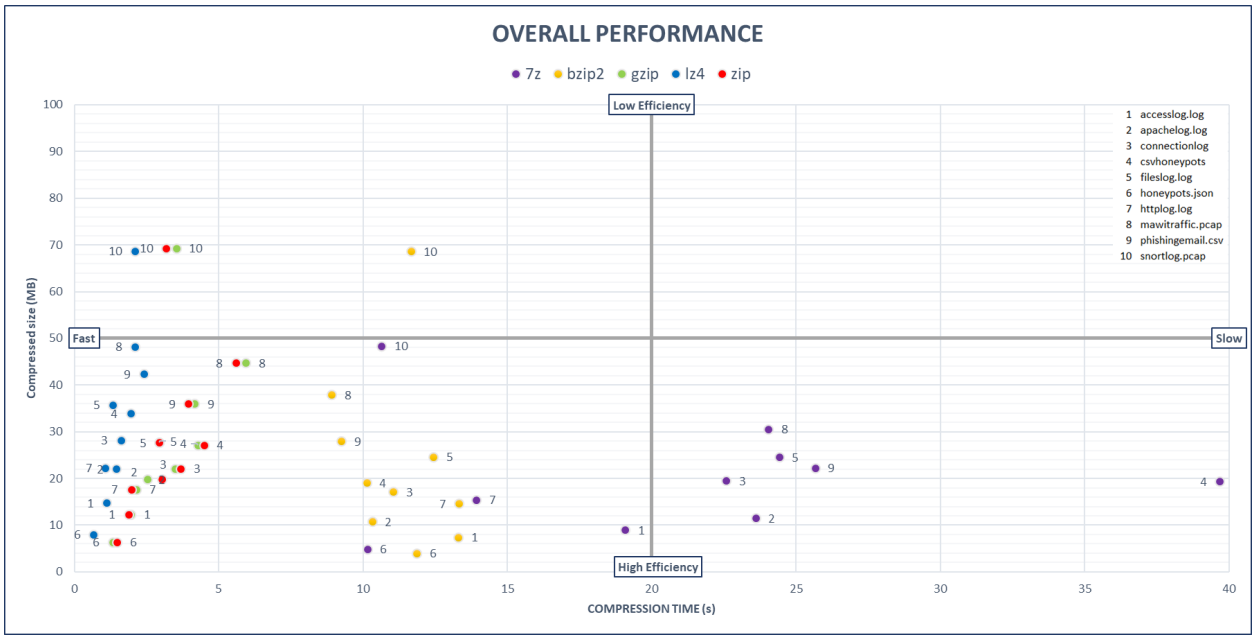

Figure 5.20: Overall performance quadrants

The next section will summarise the findings of the research and suggest some recommendation.

5.11.4 Recommendation

Compressing security log files and thereby saving time or storage space sometimes both, if there is applicable, is an essential task within every company's security management plan. Finding the fastest or the most efficient compression method however always comes with some sort of trade off. None of the tested compression algorithm can possible possess both at the same time. The following summary Table (5.8) outlines the final results. This time instead of accumulating averages or plotting graphs, a ranking table is created, that clearly represent the ranking of compression algorithm on each elements of the security log corpus.

Table 5.8: Overall performance efficiency

		Efficiency					Speed				
Nr.	**File Name**	**1st**	**2nd**	**3rd**	**4th**	**5th**	**1st**	**2nd**	**3rd**	**4th**	**5th**
1	**accesslog.log**	bzip2	7z	gzip	zip	lz4	lz4	zip	gzip	bzip2	7z
2	**apachelog.log**	bzip2	7z	gzip	zip	lz4	lz4	gzip	zip	bzip2	7z
3	**connectionlog**	bzip2	7z	gzip	zip	lz4	lz4	gzip	zip	bzip2	7z
4	**csvhoneypots**	bzip2	7z	gzip	zip	lz4	lz4	gzip	zip	bzip2	7z
5	**fileslog.log**	7z	bzip2	gzip	zip	lz4	lz4	gzip	zip	bzip2	7z
6	**honeypots.json**	bzip2	7z	gzip	zip	lz4	lz4	gzip	zip	7z	bzip2
7	**httplog.log**	bzip2	7z	gzip	zip	lz4	lz4	zip	gzip	bzip2	7z
8	**mawitraffic.pcap**	7z	bzip2	gzip	zip	lz4	lz4	zip	gzip	bzip2	7z
9	**phishingemail.csv**	7z	bzip2	gzip	zip	lz4	lz4	zip	gzip	bzip2	7z
10	**snortlog.pcap**	7z	bzip2	lz4	gzip	zip	lz4	zip	gzip	7z	bzip2
Total Ranking		**bzip2**	**7z**	**gzip**	**zip**	**lz4**	**lz4**	**zip/gzip**	**zip/gzip**	**bzip2**	**7z**

Accordingly, the most efficient by scoring 6 times out of 10 on the first place is *bzip2*, followed by *7z* scoring at the first place 4 times. *Gzip* takes a very stable third place. The last two almost head to head *zip* and *lz4*.

The speed test results concluded that *lz4* is way ahead on the first place, while *zip* and *gzip* performed equally well on the second place, just as the third place. Here some additional deeper analysis would be necessary to rectify their evaluated records. The fourth place goes undoubtedly to *bzip2*. Finally with its very slow performance *7z* closes the ranking.

Considering the compression efficiency on different file types, *bzip2* compresses ASCII and http log files (with very long lines) better, thus *7z* prefer *.pcap* files. *Gzip* has a good overall performance on all files. *Zip* and especially *Lz4* is not recommended if we want to achieve high compression performance on this file set.

Lz4 is definitely the best choice for compression speed utilisation on every file type of the corpus. *Zip* performs better than *gzip* on *.pcap* files and ASCII log files. *Json* files are mostly favoured by the *zip* compression method. Nevertheless, *Bzip2* and *7z* are not recommended to be used if compression speed at stake.

CHAPTER 6

Conclusion

6.1 Introduction

This chapter attempts to give a synopsis about the analysis has been conducted in the previous chapters. Firstly the summary of research is presented that is followed by an evaluation of the the achieved objectives. How this study can contribute to others future work is described thereafter.

6.2 Summary of Research

A statistical analysis of security log file compression methods is presented. First and foremost concerning the function of compression algorithm within the field of security log management and their role and importance in companies security policy. Chapter 1 introduced the theoretical background of this study, also highlighted the challenges concerned in connection to security log file archiving. The main research objectives and the implemented research methodology were defined as well. Chapter 2 introduced a wide variety of literature reviews that formed the backbone of this study. The history and application fields of different types of compression methods, especially their security related attributes were discussed. Chapter 3 aimed to outline the research approach and introduced the test framework and test environment as well as the five compression method that were investigated. Chapter 4 gave more insight about the security log corpus the statistical analysis was conducted on. Their different structure and major characteristics were discussed in more depth. In the first part of Chapter 5 it have been introduced the measurement criteria and all the additional data visualisation tools to define how the result were interpreted and categorised. The second part represented the actual test procedures, their outcome and evaluation that resulted in communicating our findings.

6.3 Research Objectives

The major objective of this study was to determine reasonable and effective security log data compression methods that can help maintaining security log management and capable to successfully preserve the collected data on-premises and during transmission. By observing examining the current security log file archiving procedures and best practices (2.8), security log file retention (2.9) and the applicable compression techniques (2.10) this objective was clearly outlined.

The objective to create the server environment (3.4) and downloaded the required raw samples was successfully completed. The downloaded data was unpacked, filtered and resized to finalise as our security log data corpus (4.3) for further processing. These preparation steps went smoothly.

To automate the test framework - as a more technical objective - a BASH script was constructed (3.5.1) to enable the integration of different compression algorithm. At this point the master script could definitely be simplified by applying higher programming logic, but for our purposes it was satisfying. The script went through multiple test before it has been finalised. The occurring errors were handled accordingly without any outstanding difficulties.

By conducting research on the available compression studies (2.4, 2.6), five compression method were selected (3.1) and implemented into the BASH script. The implementation must be adjusted along the test and some of the issues explained in respectively (5.6.1, 5.7.1, 5.9.1, 5.10.1). All in all the script execution and the behaviour of the compression method during runtime gave the expected results.

The resulted metrics of each test were measured, categorised and analysed accordingly (5.6, 5.7, 5.8, 5.9, 5.10). Their overall performance and evaluation were summarised in a separated section (5.11). We have taken special effort to correctly visualise the results by using different tools (5.5) to add more value to this research. By doing this a very important objective is also completed. Finally we have made some recommendation (5.11.4) what are the most fastest and most efficient way of presuming security log data.

6.4 Research Contribution

The present study attempts to address a statistical analysis of commonly available lossless compression methods on security log file data. It aims to utilise an analytical approach to determine what is the most efficient solution in terms of speed and compression ratio to preserve security logs for later examination. The firs contribution is to emphasise the importance of security log file data on an industrial level and engage company to implement the described compression methods on their daily management practices described by Ali et al. (2021), Anusooya et al. (2015), Gorge (2007).

Security log files contains sensitive data, and even though it is relatively easy to harvest, it

is more difficult to find a simple dataset that contains all sort of log data, ranging from sever log files, IDS log through network traffic capture files in one set. Therefore the second contribution is to create a collection of different type of real-life log data from legitimate source, therewith producing a unique security data corpus, that can be utilised by other researcher and compact enough to be distributed.

The third contribution is a BASH script (Appendix A) designed for security log file analysis. It is composed to enable the user for implementing different types of compression syntax that record real-time data about the performance of the implemented script during execution. It also allows the tester to define additional criteria within the script to further examine the behaviour of a given compression tool. The script has a potential for development in the field of compression tool's automation.

The fourth contribution of this thesis are the produced data metrics (Appendix A) and their analytical results that aim to support the user in decision making to find the most suitable compression method for security log data archiving. This involves to determine which data compression algorithm matches best for their requirements by considering the limitation of their security infrastructure.

We have came to a conclusion during conducting the research that by only deploying default compression method without utilising their potential can produce partially sufficient outcome in terms of efficiency and speed. A more unique approach by consequently applying the most accurate file compression algorithm (and their appropriate compression level) as it has been pointed out and automate this process can enhance to save more time and more storage space, which are either way reduce the cost of managing security log file data.

6.5 Future Work

As the concept of this work mostly builds on a wide variety of academical researches undertaken in this field mostly in connection to compression methods in general, security log file preservation techniques are not emphasised well enough.In many cases the compression dataset that a security analyst uses much large then our security dataset. Packet capture files, due to their complexity and wealth of information exchanged between endpoints, pose significant storage challenges. Hence, refining an approach to efficiently minimise them, as outlined here, could offer a promising avenue for research. Furthermore, extending our approach to larger files (ranging from tens to hundreds of gigabytes) and libraries presents an interesting experimental opportunity. Developing a script or software that scans and organises log files, providing a curated list of compression methods along with expected compression results, would be of considerable interest. Exploring the integration of AI into such a framework also shows some potential for further investigation.

References

Abel, J., & Teahan, W. (2005). Universal text preprocessing for data compression. *IEEE Transactions on Computers*, *54*(5), 497–507. https://doi.org/10.1109/TC.2005.85 (cit. on p. 8).

Abu Alsheikh, M., Lin, S., Niyato, D., & Tan, H.-P. (2016). Rate-distortion balanced data compression for wireless sensor networks. *IEEE Sensors Journal*, *16*(12), 5072–5083. https://doi.org/10.1109/JSEN.2016.2550599 (cit. on p. 8).

Abu-Taieh, E., & AlHadid, I. (2018). CRUSH: A new lossless compression algorithm. *Modern Applied Science*, *12*(11), 406. https://doi.org/10.5539/mas.v12n11p406 (cit. on p. 13).

Akbaş, E. (2023). Enhancing incident response with live logs: The significance and challenges of maintaining sufficient log retention for mitigating cyber attacks (cit. on p. 16).

Ali, A., Ahmed, M., & Khan, A. (2021). Audit logs management and security - a survey. *Kuwait Journal of Science*, *48*(3). https://doi.org/10.48129/kjs.v48i3.10624 (cit. on pp. 15, 67).

Altarawneh, H., & Altarawneh, M. (2011). Data compression techniques on text files: A comparison study. *International Journal of Computer Applications*, *26*(5), 42–54. https://doi.org/10.5120/3097-4249 (cit. on p. 14).

Anusooya, R., Rajan, J., & SatyaMurty, S. A. V. (2015). Importance of centralized log server and log analyzer software for an organization. *02*(3). https://www.irjet.net/archives/V2/i3/Irjet-v2i3365.pdf (cit. on pp. 15, 67).

Berz, D., Engstler, M., Heindl, M., Waibel, F., & Göhner, U. (2015, March 1). *Comparison of lossless data compression methods*. https://www.researchgate.net/publication/335572104_Comparison_of_lossless_data_compression_methods (cit. on p. 14).

Besedin, K. Y., Kostenetskiy, P. S., & Prikazchikov, S. O. (2015). Using data compression for increasing efficiency of data transfer between main memory and intel xeon phi coprocessor or NVidia GPU in parallel DBMS. *Procedia Computer Science*, *66*, 635–641. https://doi.org/10.1016/j.procs.2015.11.072 (cit. on p. 14).

Brezinski, D., & Killalea, T. (2002). Guidelines for evidence collection and archiving [Publisher: RFC Editor Report Number: RFC3227], RFC3227. https://doi.org/10.17487/rfc3227 (cit. on pp. 2, 4).

Capon, J. (1959). A probabilistic model for run-length coding of pictures. *IRE Transactions on Information Theory*, *5*(4), 157–163. https://doi.org/10.1109/TIT.1959.1057512 (cit. on p. 8).

Cho, K. (2000). *Tcpdpriv*. Retrieved March 23, 2024, from https://www.iijlab.net/~kjc/papers/freenix2000/node13.html (cit. on p. 33).

Chuvakin, A. (2009). *Public security log sharing*. Retrieved March 10, 2024, from https://log-sharing.dreamhosters.com/ (cit. on p. 30).

Collet, Y. (2022). *LZ4 block format description* [GitHub]. Retrieved February 10, 2024, from https://github.com/lz4/lz4/blob/dev/doc/lz4_Block_format.md (cit. on p. 21).

Crochemore, M., & Lecroq, T. (1996). Pattern-matching and text-compression algorithms. *ACM Computing Surveys*, *28*(1) (cit. on p. 9).

de Sannio, U. (2022). *USB-IDS datasets*. Retrieved March 10, 2024, from https://idsdata.ding.unisannio.it/datasets.html (cit. on p. 31).

Deutsch, L. P. (1996, May). *DEFLATE compressed data format specification version 1.3* (Request for Comments No. RFC 1951) (Num Pages: 17). Internet Engineering Task Force. https://doi.org/10.17487/RFC1951 (cit. on p. 20).

Drost, G. W., & Bourbakis, N. G. (2001). A hybrid system for real-time lossless image compression. *Microprocessors and Microsystems*, *25*(1), 19–31. https://doi.org/10.1016/S0141-9331(00)00102-2 (cit. on p. 8).

Duarte, F. (2023, March 16). *Amount of data created daily (2024)* [Exploding topics]. Retrieved January 21, 2024, from https://explodingtopics.com/blog/data-generated-per-day (cit. on p. 10).

Dubé, D., & Beaudoin, V. (2010). Lossless data compression via substring enumeration. *2010 Data Compression Conference*, 229–238. https://doi.org/10.1109/DCC.2010.28 (cit. on p. 13).

Ebrahim, M., & Mallett, A. (2018, April 19). *Mastering linux shell scripting,: A practical guide to linux command-line, bash scripting, and shell programming, 2nd edition*. Packt Publishing Ltd. (Cit. on p. 25).

ETHW. (2011). *History of lossless data compression algorithms* [ETHW]. Retrieved October 20, 2023, from https://ethw.org/File:Compression_hierarchy.png (cit. on p. 12).

Fang, W., He, B., & Luo, Q. (2010). Database compression on graphics processors. *Proceedings of the VLDB Endowment*, *3*(1), 670–680. https://doi.org/10.14778/1920841.1920927 (cit. on p. 14).

Fano, R. M. (1949). *The transmission of information* (Technical report No. 65). Massachusetts Institute of Technology. Retrieved February 12, 2024, from https://hcs64.com/files/fano-tr65-ocr-only.pdf (cit. on p. 11).

Gailly, J.-l. (2023). *GNU gzip* [GNU gzip: General file (de)compression]. Retrieved February 10, 2024, from https://www.gnu.org/software/gzip/manual/gzip.html (cit. on p. 20).

Garba, A. M., & Zirra, P. B. (2014). Analysing forward difference scheme on huffman to encode and decode data losslessly, 46–54. Retrieved January 4, 2024, from https://www.theijes.com/papers/v3-i6/Version-5/F0365046054.pdf (cit. on p. 13).

Gastegger, M. (2020). A performance comparison of 7z/LZMA and 7z/bzip2/tar. https://www.researchgate.net/publication/350049637_A_Performance_Comparison_of_7zLZMA_and_7zbzip2tar (cit. on p. 21).

Giménez, C. T., Pørez Villegas, A., & Maranon, G. A. (2010). *Http dataset CSIC 2010* [Http dataset CSIC 2010]. Retrieved March 19, 2024, from https://www.tic.itefi.csic.es/dataset/ (cit. on p. 32).

Gopinath, A., & Ravisankar, M. (2020). Comparison of lossless data compression techniques, 628–633. https://doi.org/10.1109/ICICT48043.2020.9112516 (cit. on p. 20).

Gorge, M. (2007). Making sense of log management for security purposes – an approach to best practice log collection, analysis and management. *Computer Fraud & Security*, *2007*(5), 5–10. https://doi.org/10.1016/S1361-3723(07)70047-7 (cit. on pp. 16, 67).

Greenfield, D., Wittorff, V., & Hultner, M. (2019). The importance of data compression in the field of genomics. *IEEE Pulse*, *10*(2), 20–23. https://doi.org/10.1109/MPULS.2019.2899747 (cit. on p. 9).

Hazboun, K. A., & Bassiouni, M. A. (1982). A multi-group technique for data compression. *Proceedings of the 1982 ACM SIGMOD International Conference on Management of data*, 284–292. https://doi.org/10.1145/582353.582406 (cit. on p. 3).

Holtz, K. (1993). The evolution of lossless data compression techniques [ISSN: 1095-791X]. *Proceedings of WESCON '93*, 140–145. https://doi.org/10.1109/WESCON.1993.488424 (cit. on p. 8).

Hosseini, M. (2012). A survey of data compression algorithms and their applications, 15. https://doi.org/10.13140/2.1.4360.9924 (cit. on pp. 10, 12).

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *IRE*, *40*, 1098–1101. https://doi.org/doi:10.1109/JPROC.1952.273898 (cit. on pp. 3, 12).

Hussain, A. J., Al-Fayadh, A., & Radi, N. (2018). Image compression techniques: A survey in lossless and lossy algorithms. *Neurocomputing*, *300*, 44–69. https://doi.org/10.1016/j.neucom.2018.02.094 (cit. on p. 10).

Insigthz, I. C. (2023). *Log management market size & share | analysis report | 2023-2030 | LinkedIn*. Retrieved April 27, 2024, from https://www.linkedin.com/pulse/log-management-market-size-share-analysis-report-uynwf/ (cit. on p. 16).

Intel. (2022). *What is intel® QuickAssist technology (intel® QAT)?* [Intel]. Retrieved February 8, 2024, from https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/what-is-intel-qat.html (cit. on p. 15).

Iyer, R., Jose, S., & Wilhite, D. (1994). Data compression support in databases. *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, 695–704. https://www.vldb.org/conf/1994/P695.PDF (cit. on p. 9).

Jacob, N., Somvanshi, P., & Tornekar, R. (2012). Comparative analysis of lossless text compression techniques. *International Journal of Computer Applications*, *56*, 17–21. https://doi.org/10.5120/8871-2850 (cit. on pp. 13, 14).

Jayasankar, U., Thirumal, V., & Ponnurangam, D. (2021). A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. *Journal of King Saud University - Computer and Information Sciences*, *33*(2), 119–140. https://doi.org/10.1016/j.jksuci.2018.05.006 (cit. on p. 8).

Kalajdzic, K., Ali, S. H., & Patel, A. (2015). Rapid lossless compression of short text messages. *Computer Standards & Interfaces*, *37*, 53–59. https://doi.org/10.1016/j.csi.2014.05.005 (cit. on p. 8).

Kavitha, P. (2016). A survey on lossless and lossy data compression methods. *IJCSET*, *7*(3). http://www.ijcset.com/docs/IJCSET16-07-03-049.pdf (cit. on p. 10).

Kent, K., & Souppaya, M. (2006, September 13). *Guide to computer security log management* (NIST Special Publication (SP) 800-92). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-92 (cit. on pp. 4, 15).

Kim, G. Y., Paik, J.-Y., Kim, Y., & Cho, E.-S. (2022). Byte frequency based indicators for crypto-ransomware detection from empirical analysis. *Journal of Computer Science and Technology*, *37*(2), 423–442. https://doi.org/10.1007/s11390-021-0263-x (cit. on p. 41).

Kim, J., & Cho, J. (2019). Hardware-accelerated fast lossless compression based on LZ4 algorithm. *Proceedings of the 2019 3rd International Conference on Digital Signal Processing*, 65–68. https://doi.org/10.1145/3316551.3316564 (cit. on p. 21).

Kolo, J. G., Shanmugam, S. A., Lim, D. W. G., Ang, L.-M., & Seng, K. P. (2012). An adaptive lossless data compression scheme for wireless sensor networks. *Journal of Sensors*, *2012*, 20. https://doi.org/10.1155/2012/539638 (cit. on p. 8).

Langiu, A. (2013). On parsing optimality for dictionary-based text compression—the zip case. *Journal of Discrete Algorithms*, *20*, 65–70. https://doi.org/10.1016/j.jda.2013.04.001 (cit. on p. 20).

Lee, K.-H., Slattery, O., Lu, R., Tang, X., & McCrary, V. (2002). The state of the art and practice in digital preservation. *Journal of Research of the National Institute of Standards and Technology*, *107*(1), 93–106. https://doi.org/10.6028/jres.107.010 (cit. on p. 4).

Lelewer, D. A., & Hirschberg, D. S. (1987). Data compression. *ACM Computing Surveys*, *19*(3), 261–296. https://doi.org/10.1145/45072.45074 (cit. on p. 11).

Leonidas, M. (2019, December 26). *Log retention solutions* [NIC inc.]. Retrieved February 3, 2024, from https://www.nicitpartner.com/log-retention-solutions/ (cit. on p. 16).

Li, S.-H., Yen, D. C., & Chuang, Y.-P. (2016). A real-time audit mechanism based on the compression technique. *ACM Transactions on Management Information Systems*, *7*(2), 4:1–4:25. https://doi.org/10.1145/2629569 (cit. on p. 17).

LogicMonitor. (2022, March 15). *What is log retention?* [LogicMonitor]. Retrieved February 3, 2024, from https://www.logicmonitor.com/blog/what-is-log-retention (cit. on p. 16).

Marjai, P., Lehotay-Kéry, P., & Kiss, A. (2022). A novel dictionary-based method to compress log files with different message frequency distributions. *Applied Sciences*, *12*(4), 2044. https://doi.org/10.3390/app12042044 (cit. on p. 17).

Mark, L., Wagner, A., Boschi, E., Zseby, T., & Trammell, B. (2009, October). *Specification of the IP flow information export (IPFIX) file format* (Request for Comments No. RFC 5655) (Num Pages: 64). Internet Engineering Task Force. https://doi.org/10.17487/RFC5655 (cit. on p. 20).

Marr, B. (2024). *How much data do we create every day? the mind-blowing stats everyone should read | bernard marr*. Retrieved February 6, 2024, from https://bernardmarr.com/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/ (cit. on p. 10).

Mittal, S., & Vetter, J. S. (2015). A survey of architectural approaches for data compression in cache and main memory systems. *IEEE Transactions on Parallel and Distributing Systems*, *27*, 1524–1536. https://doi.org/10.1109/TPDS.2015.2435788 (cit. on p. 14).

Mohammed, A.-l., & Emary, I. (2007). Comparative study between various algorithms of data compression techniques. *Lecture Notes in Engineering and Computer Science*, *2167*. https://www.researchgate.net/publication/44261387_Comparative_Study_between_Various_Algorithms_of_Data_Compression_Techniques (cit. on pp. 3, 10).

Mohr, R., & Henderson, T. C. (1986). Arc and path consistency revisited. *Artificial Intelligence*, *28*(2), 225–233. https://doi.org/10.1016/0004-3702(86)90083-4 (cit. on p. 13).

Nemerson, E. (2015). *Squash compression benchmark* [Squash compression benchmark]. Retrieved January 7, 2024, from http://quixdb.github.io/squash-benchmark/ (cit. on p. 15).

Nielsen, L. H. (2023, September 13). Phishing email curated datasets. https://doi.org/10.5281/zenodo.8339691 (cit. on p. 31).

Oswal, S., Singh, A., & Kumari, K. (2016). Deflate compression algorithm. (1). Retrieved February 2, 2024, from http://pnrsolution.org/Datacenter/Vol4/Issue1/58.pdf (cit. on p. 21).

Otten, F., Irwin, B., & Thinyane, H. (2007). Evaluating compression as an enabler for centralised monitoring in a next generation network, 6. https://api.semanticscholar.org/CorpusID:209440835 (cit. on p. 3).

Ozsoy, A., Swany, M., & Chauhan, A. (2014). Optimizing LZSS compression on GPGPUs. *Future Generation Computer Systems*, *30*, 170–178. https://doi.org/10.1016/j.future.2013.06.022 (cit. on p. 15).

Pavlov, I. (2024). *7z format* [7z format]. Retrieved February 10, 2024, from https://www.7-zip.org/7z.html (cit. on p. 21).

Pu, I. M. (2006). *Fundamental data compression*. Butterworth-Heinemann. (Cit. on p. 8).

Radley, J. J. (2015). *Pseudo-random access compressed archive for security log data* [Master thesis]. Rhodes University [Publisher: Rhodes University; Faculty of Science, Computer Science]. Retrieved September 16, 2023, from https://vital.seals.ac.za/vital/access/manager/Repository/vital:4723?site_name=GlobalView&view=null&f0=sm_creator%3A%22Radley%2C+Johannes+Jurgens%22&sort=null (cit. on p. 4).

Rahman, M. A., & Hamada, M. (2019). Lossless image compression techniques: A state-of-the-art survey [Number: 10 Publisher: Multidisciplinary Digital Publishing Institute]. *Symmetry*, *11*(10), 1274. https://doi.org/10.3390/sym11101274 (cit. on p. 10).

Rahman, M. A., & Hamada, M. (2020). Burrows–wheeler transform based lossless text compression using keys and huffman coding. *Symmetry*, *12*(10), 1654. https://doi.org/10.3390/sym12101654 (cit. on p. 13).

Reghbati, H. (1981). Special feature an overview of data compression techniques. *Computer*, *14*(4), 71–75. https://doi.org/10.1109/C-M.1981.220416 (cit. on p. 14).

Rinnan, R. (2005). *Benefits of centralized log file correlation* [Master thesis]. Gjøvik University College [Accepted: 2008-04-01T08:35:43Z]. Retrieved November 17, 2023, from https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/143787 (cit. on p. 16).

Ruxanayasmin, B., Ananda Krishna, B., & Subhashini, T. (2013). Implementation of data compression techniques in mobile ad hoc networks. *International Journal of Computer Applications*, *80*(8), 8–12. https://doi.org/10.5120/13879-1764 (cit. on p. 8).

Salomon, D. (2002, February 8). *A guide to data compression methods* [Google-Books-ID: 8UkKN7cKllsC]. Springer Science & Business Media. (Cit. on pp. 7, 11).

Salomon, D. (2007, December 18). *A concise introduction to data compression*. Springer Science & Business Media. (Cit. on p. 8).

Sardar Ali, M. (2021). Features of application of data compression methods. *6*, 969–983. https://doi.org/10.25212/lfu.qzj.6.3.34 (cit. on p. 10).

Sayood, K. (2002, December 18). *Lossless compression handbook*. Elsevier. (Cit. on pp. 8, 10).

Sayood, K. (2017, October 23). *Introduction to data compression* (5th). Morgan Kaufmann. (Cit. on pp. 2, 3, 8).

Schmidt, K., Phillips, C., & Chuvakin, A. (2012, December 31). *Logging and log management: The authoritative guide to understanding the concepts surrounding logging and log management* [Google-Books-ID: Rf8M_X_YTUoC]. Newnes. (Cit. on p. 4).

Sconzo, M. (2015). *Security-data-analysis* [original-date: 2015-02-04T22:16:16Z]. Retrieved April 4, 2024, from https://github.com/sooshie/Security-Data-Analysis (cit. on p. 30).

Sconzo, M. (2024). *SecRepo - security data samples repository* [SecRepo.com - samples of security related data]. Retrieved March 9, 2024, from http://www.secrepo.com/ (cit. on pp. 30, 32).

Semunigus, W., & Pattanaik, B. (2021). Analysis for lossless data compression algorithms for low bandwidth networks. *Journal of Physics: Conference Series*, *1964*, 042046. https://doi.org/10.1088/1742-6596/1964/4/042046 (cit. on p. 14).

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*(3), 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x (cit. on p. 11).

Sharif, A. (2022). *Log files: Definition, types, and importance - CrowdStrike* [Crowdstrike.com]. Retrieved April 4, 2024, from https://www.crowdstrike.com/cybersecurity-101/observability/log-file/ (cit. on p. 28).

Singh, A. V., & Singh, G. (2012). A survey on different text data compression techniques. *International Journal of Science and Research (IJSR)*, *3*(7). https://www.ijsr.net/archive/v3i7/MDIwMTQxMjk4.pdf (cit. on p. 11).

Skibinski, P., & Swacha, J. (2007). Fast and efficient log file compression. *325*, 56–69. https://ceur-ws.org/Vol-325/paper06.pdf (cit. on p. 17).

Smith, C. A. (2010). A survey of various data compression techniques. Retrieved February 6, 2024, from https://www.hsc.edu/documents/academics/mathcs/pendergrass/honorspaper_alexsmith.pdf (cit. on p. 11).

Söderström, O., & Moradian, E. (2013). Secure audit log management. *Procedia Computer Science*, *22*, 1249–1258. https://doi.org/10.1016/j.procs.2013.09.212 (cit. on p. 16).

Souley, B., Das, P., & Tanko, S. (2014). A comparative analysis of data compression techniques. *International Journal of Applied Science and Engineering*, *2*(1), 63. https://doi.org/10.5958/2322-0465.2014.01118.6 (cit. on p. 10).

Sourceforge. (2022, January 6). *Bzip2* [SourceForge]. Retrieved February 10, 2024, from https://sourceforge.net/projects/bzip2/ (cit. on p. 20).

Spillner, J. (2020). Comparison and model of compression techniques for smart cloud log file handling. *2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)*, 1–6. https://doi.org/10.1109/CCCI49893.2020.9256609 (cit. on p. 17).

Srisooksai, T., Keamarungsi, K., Lamsrichan, P., & Araki, K. (2012). Practical data compression in wireless sensor networks: A survey. *Journal of Network and Computer Applications*, *35*(1), 37–59. https://doi.org/10.1016/j.jnca.2011.03.001 (cit. on p. 8).

Stathopoulos, V., Kotzanikolaou, P., & Magkos, E. (2008). Secure log management for privacy assurance in electronic communications. *Computers & Security*, *27*(7), 298–308. https://doi.org/10.1016/j.cose.2008.07.010 (cit. on p. 16).

Statista. (2024). *Big data analytics market size worldwide 2029* [Statista]. Retrieved February 6, 2024, from https://www.statista.com/statistics/1336002/big-data-analytics-market-size/ (cit. on p. 9).

Tariq, J., Mosleh, M. F., Abdulameer, M., Obeidat, H. A., & Obeidat, O. A. (2023, March 1). *Hybrid lossless compression techniques for english text*. Retrieved February 10, 2024, from https://journal.mtu.edu.iq/index.php/MTU/article/view/1059 (cit. on p. 21).

Taylor, P. (2023, August 23). *Data growth worldwide 2010-2025* [Statista]. Retrieved September 19, 2023, from https://www.statista.com/statistics/871513/worldwide-data-created/ (cit. on p. 1).

To, H., Chiang, K., & Shahabi, C. (2013). Entropy-based histograms for selectivity estimation. *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 1939–1948. https://doi.org/10.1145/2505515.2505756 (cit. on p. 41).

Tugnarelli, M. D., Fornaroli, M. F., Santana, S. R., Jacobo, E., & Díaz, J. (2018). Analysis of methodologies of digital data collection in web servers [Series Title: Communications in Computer and Information Science]. In A. E. De Giusti (Ed.), *Computer science – CACIC 2017* (pp. 265–271, Vol. 790). Springer International Publishing. https://doi.org/10.1007/978-3-319-75214-3_25 (cit. on pp. 1, 4).

Welch. (1984). A technique for high-performance data compression. *Computer*, *17*(6), 8–19. https://doi.org/10.1109/MC.1984.1659158 (cit. on p. 12).

Wilbert, B., & Chen, L. (2012). Log management and retention in corporate environments. *Proceedings of the International Conference on e-Learning, e-Busines*, 5. Retrieved February 2, 2024, from http://worldcomp-proceedings.com/proc/p2012/EEE3419.pdf (cit. on p. 16).

WRCCDC. (2021). *WRCCDC public archive* [Western regional collegiate cyber defense competition]. Retrieved March 9, 2024, from https://archive.wrccdc.org/pcaps/2021/ (cit. on p. 33).

Yang, P., Xiong, N., & Ren, J. (2020). Data security and privacy protection for cloud storage: A survey. *IEEE Access*, *8*, 131723–131740. https://doi.org/10.1109/ACCESS.2020.3009876 (cit. on p. 4).

Yao, K., Sayagh, M., Shang, W., & Hassan, A. E. (2022). Improving state-of-the-art compression techniques for log management tools. *IEEE Transactions on Software Engineering*, *48*(8), 2748–2760. https://doi.org/10.1109/TSE.2021.3069958 (cit. on p. 17).

Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Trans. Inform. Theory*, *23*(3), 337–343. https://doi.org/10.1109/TIT.1977.1055714 (cit. on p. 12).

APPENDIX A

Datasheet

This appendix provides a direct link

Bachelor Project NUC - AN ANALYSIS OF COMPRESSION METHODS FOR SECURITY DATA

to the following:

- Security data corpus
  1. accesslog.log
  2. apachelog.log
  3. connectionlog
  4. csvhoneypots
  5. fileslog.log
  6. honeypots.json
  7. httplog.log
  8. mawitraffic.pcap
  9. phishingemail.csv
  10. snortlog.pcap
- BASH script
  1. compress-decompress-7z.sh
  2. compress-decompress-bzip2.sh
  3. compress-decompress-gzip.sh
  4. compress-decompress-lz4.sh
  5. compress-decompress-zip.sh
- Datasheet with statistical data
  1. Stat-Summary_FINAL_28.04.2024.xlsx

The security data files are final as well as the datasheet. The five BASH scripts have slightly been improved along our experiment. The easiest way to execute the script is to place the same directory as the security data corpus is, alternatively follow the instructions:

1. Create a testing directory with *mkdir directory_name* command.
2. Copy the script into the testing directory.
3. Open the script with *sudo nano file_name* command.
4. Modify the execution path where the security data corpus is located and save the file by CTRL+X.
5. Run the script on the command line.

Word count metrics

NUC Bachelor Project Word Count:
Total Sum count: 22667 Words in text: 22191 Words in headers: 325 Words outside text (captions, etc.): 143 Number of headers: 120 Number of floats/tables/figures: 48 Number of math inlines: 2 Number of math displayed: 6
NOTE: References are excluded.