Optimizing of Multi-Layer Perceptron for Detecting PDF Malware: A study in Machine Learning

by

Muhammad Saad Ali

A thesis submitted to the faculty of Information Security Department, Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfillment of the requirements for the degree of MS in Information Security

May 2023

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS Thesis written by **Mr. Muhammad Saad Ali**, Registration No. **0000319774**, of **Military College of Signals** has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations/MS Policy, is free of plagiarism, errors, and mistakes and is accepted as partial fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members and local evaluators of the scholar have also been incorporated in the said thesis.

Signature: _____

Name of Supervisor

Assoc Prof Dr. Muhammad Faisal Amjad

Date: _____

Signature (HOD): _____

HoD
Information Security
Military College of Sigs

Date: _____

Signature (Dean/Principal) _____

Date: 5/7/23

Brig
Dean, MCS (NUST)
(Asif Masood, Phd)

Declaration

I hereby declare that no portion of work presented in this thesis has been submitted in support of another award or qualification either at this institution or elsewhere.

_____

MS Student

Dedication

“In the name of Allah, the most Beneficent, the most Merciful"

I dedicate this thesis to my mother, brother and teachers who supported me in each step of the way.

Acknowledgments

All praises to Allah for the strengths and his blessing in completing this thesis.

I would like to convey my gratitude to my supervisor, Dr. Faisal Amjad for his supervision and constant support. His invaluable help of constructive comments and suggestions throughout the experimental and thesis works are major contributions to the success of this research. Also, I would thank my committee members; Assoc Prof Dr. Mian Muhammad Waseem Iqbal, and Asst Prof Dr. Fawad Khan for their support and knowledge regarding this topic.

Last, but not the least, I am highly thankful to my mother and brother. They have always stood by my dreams and aspirations and have been a great source of inspiration for me. I would like to thank them for all their care, love and support through my times of stress and excitement.

Abstract

In this thesis, we demonstrate that multi-layer perceptrons (MLPs) are a promising approach for detecting PDF-based malware. Malware in the form of PDF files is becoming increasingly prevalent, making it crucial to develop effective detection methods. Traditional methods for detecting malware, such as signature-based detection, are becoming less effective as attackers can easily evade them by modifying the malicious code. To train our MLP, we first collected a large dataset of both benign and malicious PDFs. The dataset was pre-processed to extract relevant features, such as the presence of certain keywords and the structure of the PDF file. In total we used 37 static representative features. We used a combination of supervised learning techniques to train the MLP on this dataset. The trained model was then evaluated on a separate test dataset and was shown to have high accuracy of about 96% in detecting PDF-based malware. We also investigated the effect of different feature selection methods and the impact of network architecture on the performance of the model. The results demonstrate that using MLPs for detecting PDF-based malware is an effective approach and can achieve high accuracy. Moreover, we also proposed an approach to increase the robustness of the model by using adversarial machine learning techniques to improve the model's ability to detect novel and evasive malware. In conclusion, this thesis presents a novel approach for training MLPs to detect PDF-based malware, and the results demonstrate the effectiveness of this approach. The proposed approach could be used to improve the security of systems that handle PDF files and provide a new tool for the security community to fight against PDF-based malware.

Table of Contents

Page

Chapter

List of Figures

List of Tables

Appendix A

Appendix B

Appendix C

Chapter 1

Introduction

1.1 Overview

PDF, which stands for Portable Document Format, was invented in 1991 by Adobe Systems co-founder Dr. John Warnock. The first version of the PDF specification, version 1.0, was released in 1993. Since then, PDF has become one of the most popular file formats for digital documents, as it allows users to share documents across different platforms and devices while preserving the original formatting and layout.

It was made an open standard in 2008, as ISO 32000–1: 2008 [1]. According to this standard PDF at its core is an advanced imaging model which is derived from PostScript. PDF allows device-independent and resolution-independent viewing of a document. But unlike the programming language Postscript, PDF is a structured binary file format that is optimized for high performance in interactive viewing. PDF/X (ISO 15930) for printing, PDF/A (ISO 19005) for archiving and PDF/E (ISO 24517) for engineering documents are specific standards for PDF applications. A basic structure of a PDF can be observed from the tree figure below.

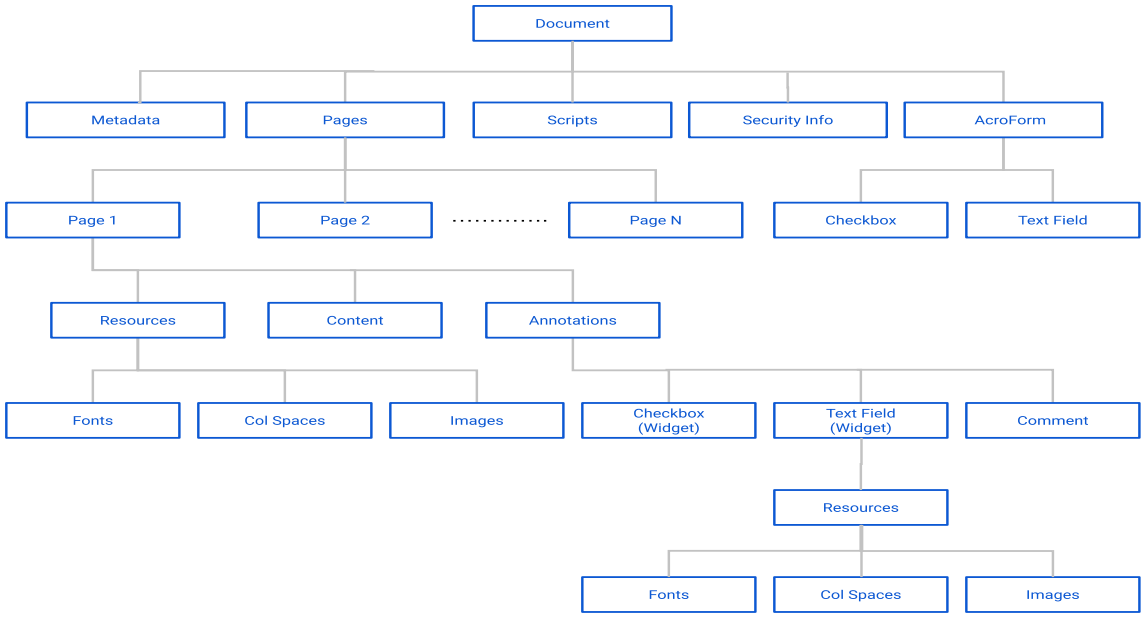

Figure 1: Tree structure of a PDF

Apart from Adobe software. PDFs can be modified with the help of Microsoft office which first converts it to word document and then edit it. But it's not usually a very convenient conversion.

Libre Office Draw is an open-source alternative to Adobe software for editing a PDF. But then again it has flaws in it which make it difficult to produce an exact copy of the PDF after editing it. Adobe Acrobat Pro has an advantage over other software as it is made by Adobe itself and can produce exact PDF with all the exact fonts.

They invented it because they wanted a format that can be used by any device or software. This is the reason why today we can open a PDF file even in our browsers. It has been built to look the same in any device or software. PDFs can contain fonts, embedded images, hyperlinks, interactive buttons, videos, images and text. A raw PDF when opened with note pad shows something like in following figure. It has head, body, xref table and trailer. When updated a new body, xref table and trailer is added.

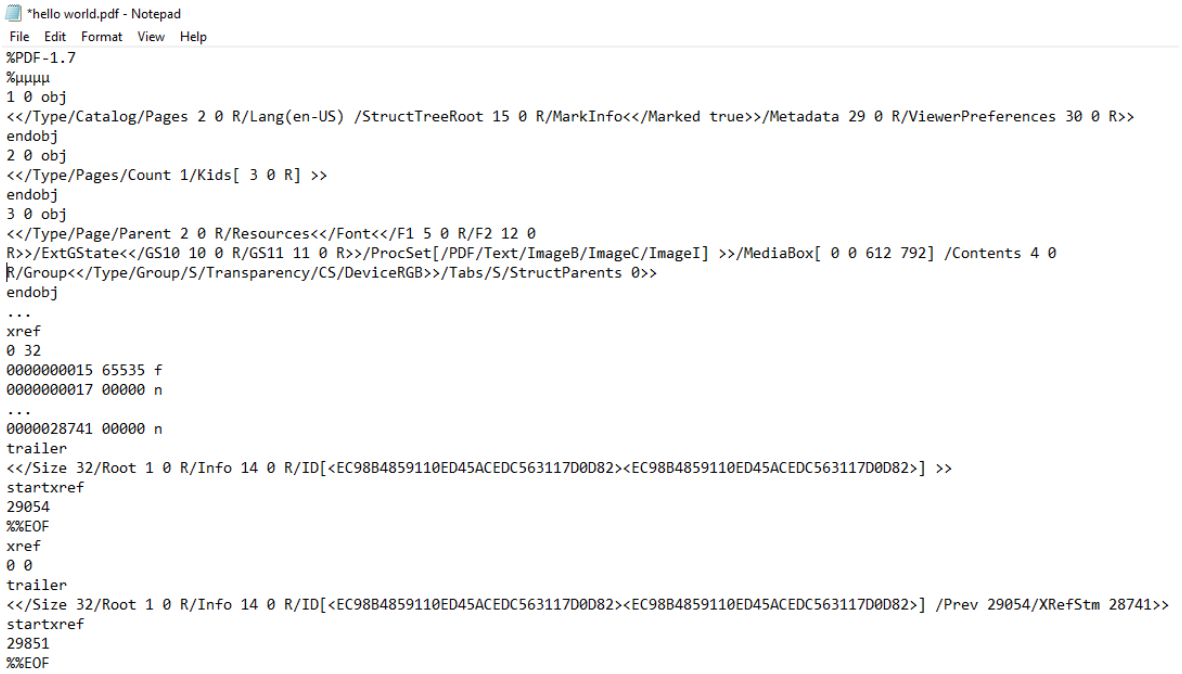

Figure 2: Raw PDF

PDFs also allow users to encrypt themselves to protect themselves from unauthorized access. The algorithms used by PDFs are RC4 and AES.

The detection of malware is a critical task in cybersecurity. Malware can cause significant harm, including data loss, financial theft, and network downtime. In recent years, machine learning algorithms have been increasingly used for malware detection due to their ability to learn from patterns in data and detect previously unseen malware. One such algorithm is the Multi-Layer Perceptron (MLP), a type of artificial neural network that has shown great promise in detecting malware.

PDF files have become a popular medium for the distribution of malware due to their ability to contain a variety of content such as images, videos, and interactive forms. PDF malware can be embedded in a document in various ways, such as using JavaScript, embedded objects, or exploiting vulnerabilities in PDF readers. The detection of PDF malware is a challenging task due to the complexity and variety of techniques used to hide malicious content.

In this thesis, we will explore the optimization of MLP for detecting PDF malware. We will begin by discussing the features extraction for PDF files, followed by an overview of architecture and training of MLPs for PDF files. We will then discuss the steps involved in optimizing the MLP, including dataset curation, hyperparameter tuning, and model evaluation. Finally, we will discuss some challenges and future directions for optimizing MLP for detecting PDF malware.

1.2 Motivation and Problem Statement

Malware in the form of PDF files is becoming increasingly prevalent because of their versatility as a document type and their robustness as unchangeable. These are the reasons why PDFs have become very popular.

With the increase in PDF files popularity, malwares in the PDF files have also become popular. This is why it is crucial to develop effective detection methods for these malwares.

Traditionally PDF malware just like any other malware was detected through an Antivirus program which heavily relies on signature-based detection to detect them [2]. According to this method a PDF malware is compared with a list of malware signatures to declare it as malicious or benign. [3]

The problem with signature-based detection is that malware authors are using these signature libraries while creating malwares which practically makes these Anti-Virus programs useless for detecting PDF malwares and malwares in general.[4] [5]

Hence, there is a need for an effective detection scheme. After detection of an attacker, the utmost requirement is of a mechanism which enables a node to do its effective defense.

Machine learning can provide a better mathematical framework for analyzing these security issues in the network.

1.3 Objectives

Following are the objectives of this research:

- To conduct a study of various obfuscation techniques
- To understand working of windows PDF malwares and their countermeasures

- To train an artificial intelligence model to detect PDF malware.

1.4 Thesis Contribution

To the best of our knowledge the Multi-Layer Perceptron method proposed in this paper has not been used for training over the dataset.

The main contributions of this work are as follows:

- We have provided a deep understanding of PDF malware that can be detected using different features of a PDF file.
- We propose a MLP based Artificial Intelligence model for training a dataset of PDF malware.

1.5 Thesis Organization

The thesis is structured as follows:

- Chapter 1 provides an overview of the whole thesis, provides introduction, motivation, objectives, contribution, organization and literature review.
- Chapter 2 contains the features information and details. It discusses features selection, general features and structural features.
- Chapter 3 discusses the evaluation parameters for the proposed algorithms for its comparison with rest of the algorithms and results obtained by the experiment.
- Chapter 4 marks the end of the document. The conclusion, challenges and future work areas are revealed in this chapter.

1.6 Literature Review

The dataset that we are using is a relatively new dataset. It was published in the February of 2022 [6] therefore there is not a lot of work done on it. [7] Proposed a new technique namely Invasive Weed Optimization with Stacked Long Short-Term Memory (IWO-S-LSTM). To test the outcomes of this technique, two datasets were used, one of which is CIC-Evaisve-PDFMal2022. [8] used CIC-Evasive-PDFMal2022 to train a variant of a standard detector and found out that predictions of maliciousness had improved.[9] Improved the robustness of machine learning based PDF malware detection systems when trained with CIC-Evasive-

PDFMal2022. The algorithm used by [9] was Random Forest and accuracy obtained was 99.65% with an F1 score of 99.7%. [10] uses AdaBoost decision tree to train a model over CIC-Evasive-PDFMal2022. Accuracy achieved was 98.84% and a prediction interval of 2.174 microseconds.[11] proposed a new attack called deep reinforcement learning (DRL)—based attack framework to make PDF files undetectable. Simultaneously a DRL-based defense strategy is proposed to counter this attack. Their results show that both this attack is effective as well as its mitigation strategy.[12] used 17 datasets for malware classification to train a model based on One Rule which is basically a decision tree with a root nod. Although it has been accepted in the paper that this One Rule model is not very effective against so many features which are present in CIC-Evaisve-PDFMal2022.

[13] believes that static analysis of malwares is better then dynamic analysis which is resource intensive. They have given an account of research done on PE32 which was first released in 1992 in windows 3.1. The reason they have taken PE32 and not PE64 is backward integration. This means that a 64 bit architecture system is made to run as such that it can also run 32 bit architecture software. But a 32 bit architecture system cannot run 64 bit software. The paper claims that most of the malware is 32 bit because it can be generalized and effect a diverse range of systems. The paper also gives an account of machine learning tools like Weka or Waikato Environment, Python weka wrapper, LIBSVM, RapidMiner and Dlib which can be used to deploy machine learning algorithms. The paper provides a history of different machine learning algorithms which are 24 in total from literature published from 2004 onwards but none use neural networks for training their models. This paper also provides an account of the features selection as observation of ShortInfo_Directories, ShortInfo_Xor, ShortInfo_FileSize, ShortInfo_Detected, ShortInfo_Sections, DigitalSignature, Packer, AntiDebug, AntiVM, SuspiciousAPI, SuspiciousSections and Url from PE32 header as well as Byte n-Gram, Opcode n-gram and API calls from the code. For their experiment, authors have created their own dataset of benign files while they have taken malicious files from VXheaven [14] has now been renamed vx-underground)and VirusShare [15]. PEFRAME, HEXDUMP and OBJDUMP are the tools used for feature extraction. In their own experimental setup they claim to use neural networks using fivefold cross validation technique but unfortunately have not provided any stats on it. The highest accuracy achived was 95.53 by C4.5 algorithm.

[16] introduced controlled flow graphs, behavioral features like performance metrics, memory information and system calls. It is claimed in this paper that todays modern malware incorporate polymorphism and change themselves on the go therefore it is static malware analysis is of not much use. This paper shows the use of cuckoo sandbox for generating dynamic analysis report.

[17] points out that there are fundamental differences between intrusion detection and other applications. There are 6 key differences. First one is that machine learning is generally better at finding similarities not differences. Then there is a very high cost of classification error. After that there is a semantic gap between detection results and their operational interpretability. There is also a huge variability in perceiving what is normal. It is also observed that there is difficulty in sound evaluation of the result. And finally operating in an adversarial setting is something that differentiates intrusion detection with from other applications. Also, in the same paper it is claimed that most organizations keep their malware data proprietary and don't share it. Open-source datasets of malwares are created by combining known malwares in the wild that are tagged as malicious by multiple accounts. This means that these datasets are not optimal and may not be able to detect even known proprietary malwares [18, 19]. There has been some work done to tag even slightly malicious portable executables as malicious to increase security and train the models better. [20,21] The datasets mentioned in this paper are from VX heaven [14], VirusShare [15], MalImg [22], MS malware Classification[23], EMBER [24], MalRec [25], Malware Training Sets [26], Mal-API-2019 [27], the Zoo (malware DB) [28], Virus Total [29] and Meraz'18 Kaggle [30]. It is claimed in this paper that most often the features used are simple features like metadata which result in high success rate of more then 98 percent leaving little room for improvement [31]. Resources like MalImg [22] uses deep learning to detect maliciousness of a program. How this works is by first extracting the binary of the program and then converting that binary of zeros and ones into RGB files. This produces an image which is completely not understandable by humans but is understandable computer. By using deep learning computers can identify the patterns in the file and when trained over a large dataset, might be able to identify new malwares. Though even this technique is not completely fool proof but given the fact that malware authors tend to innovate and come up with newer malwares every here and there, this type of model is better suited for unknown and novel malwares and hence better suited for deploying in IDSs and IPSs. However there is a problem with this type of machine learning application and that is high computational resource

requirement. Image processing for machine learning algorithms require a lot of computer processing power and often require GPUs (Graphical Processing units), IPU (dedicated image processing units) and TPUs (Tensor Processing units) [32] Although google does provide these resources to be used by general public but image processing is still resource intensive [33]To convert a program to binary and then to image is just too much work for a result that can be achieved by applying neural networks on the program themselves. This saves computational power and provides almost identical results. [34] used whole binary of a PE and directly fed it into the neural network.

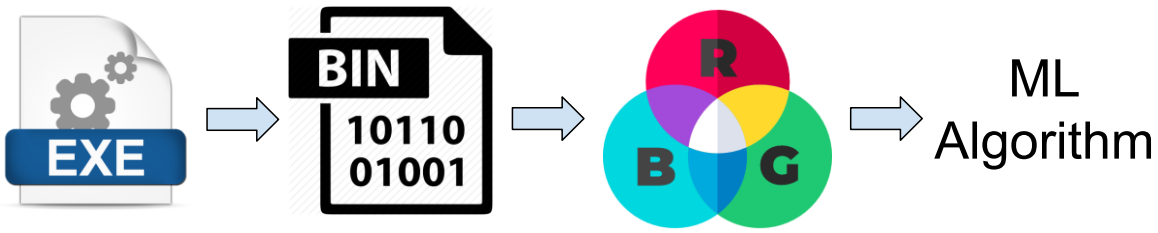

Figure 3: Training neural network with RGB

EMBER [2] is a dataset known for 1 percent false positive rate. The features having highest impact on EMBER [35] are general file information from the PE header such as virtual size of the file, thread local storage, resources, as well as the file size and number of symbols; header information from the COFF header providing the timestamp, the target machine, linker versions, and major and minor image versions; import functions obtained by parsing the address table; exported functions; section information including the name, size, entropy virtual size and list of strings representing section characteristics; byte histogram representing the counts of each byte value; byte-entropy histogram approximating the joint distribution of entropy and a given byte value; simple statistics about printable strings that are at least five characters long. Specifically providing information on strings that begin with “C:\", “http://", “https://" or “HKEY_". Over all the features represented in this paper are byte counts (BYTE), the size of the hexadecimal representation and the address of the first byte sequence (MD1),byte entropy (ENT), image representation using Haralick features (IMG1), Local

Binary Patterns (IMG2), histogram of the length of strings extracted from the hexadecimal file (STR), the size of the number of line in the disassembled file (MD2), the frequency of a set of symbols in the disassembled file (-, +, *,], [, ?, @) (SYM), the frequency of the occurrence of a subset of 93 of possible operation codes in the disassembled file (OPC), the frequency of the use of registers (REG), the frequency of the use of the top 794 Window API calls, characteristics of the sections in the binary (SEC), statistics around using db, dw, and dd instructions which are used for setting byte, word, and double word and are used to obfuscate API calls (DP), the frequency of 95 manually chosen keywords from the disassembled code (MISC)

[17] claims behavioral or dynamic analysis as active field of research. They have tried to use behavioral analysis of malware to detect maliciousness. For this purpose, they have leveraged MITRE attack framework. This is a framework that outlines all the steps of an attack. The authors have utilized MITRE malware behavior catalog or MBC to try to capture dynamic behavior of a malware. The attribute that come under anti-behavioral analysis are debugger detection, debugger evasion, dynamic analysis evasion, emulator evasion, memory dump evasion, sandbox detection, executable code virtualization, virtual machine detection, conditional execution and capture evasion. They have also incorporated Microsoft Malware Classification Challenge which has distinguished malware families as Ramnit (worm), Lollipop (Adware), Kelihos_ver3 (backdoor),Vundo (Trojan), Simda (backdoor), Tracur (trojan downloader), Kelihos_ver1 (backdoor), Obfuscator.ACY (obfuscated malware) and Gatak (backdoor). Apart from MBC and Microsoft malware classification challenge, authors have also mentioned Semantic Malware Attribute Relevance Tagging or SMART. SMART provides a richer set of technical features for malware analysis. On VX Heaven, PE-Miner [68] achieves a detection rate greater than 99% only using structural information (PE and section header information), DLLs and object files. Such statistics are common in literature where high is being achieved but one has to understand this that machine learning task should be to detect malware and not just to identify behaviors.

Chapter 2

Features

2.1 Overview

Feature selection and extraction from Files is a critical step in optimizing MLPs for detecting PDF malware. The goal of feature extraction is to transform the input data into a set of features that can be used to train the MLP. In the case of PDF files, feature extraction involves extracting information from the document's content, structure, and metadata. There are various feature extraction techniques that can be used for PDF files, such as those based on textual analysis, image processing, and metadata analysis.[36]

Text-based features can be extracted by converting the PDF document to plain text and then applying techniques such as bag-of-words, n-grams, or term frequency-inverse document frequency (TF-IDF). These techniques involve representing the document as a vector of features based on the frequency of occurrence of individual words or combinations of words in the text. Text-based features can be effective in detecting PDF malware that contains textual content, such as phishing attacks or social engineering scams.[37]

Image-based features can be extracted by converting the PDF document to images and then applying techniques such as edge detection, texture analysis, or color analysis. These techniques involve representing the document as a vector of features based on the visual characteristics of the images in the document. Image-based features can be effective in detecting PDF malware that contains images, such as those used in steganography or obfuscation.[37]

Metadata-based features can be extracted by analyzing the metadata of the PDF document, such as the author, title, creation date, and modification date. These features can be used to identify patterns in the metadata that are indicative of malicious activity, such as documents with suspicious authors or creation dates. Metadata-based features can be effective in detecting PDF malware that is designed to exploit vulnerabilities in PDF readers or manipulate the document's metadata.

For the purpose of this thesis, we have selected 12 general features and 25 structural features of PDF, details of which are given below.

2.2 General Features

General features are the features of a PDF file which are not generally related to its maliciousness directly but can give an idea about the covert maliciousness of a PDF when they are not exactly how they are supposed to be. For this thesis, we have decided to use 12 such features, details of which are in the following paragraphs.

2.2.1 Size

One method for detecting PDF malware is by analyzing the size of the PDF file. PDF files that are abnormally large or small may be an indicator of malicious activity. In this part, we will explore the use of PDF size for PDF malware detection.

PDF malware is a type of malware that is embedded within a PDF file. The malware can be designed to exploit vulnerabilities in the PDF reader software or to trick the user into downloading and executing the malware. PDF malware can be used to steal information, compromise systems, and launch further attacks.

PDF files can contain various types of malwares, such as viruses, trojans, and ransomware. These malicious files can be hidden within the PDF file, or they can be disguised as legitimate documents.[8]

PDF files can vary in size depending on the content of the file. A basic PDF file with no images or graphics may be only a few kilobytes in size, while a PDF file with high-quality images, videos, and other multimedia content can be several megabytes in size. In general, the size of a PDF file is determined by the amount of data it contains.

The size of a PDF file can also be affected by compression. PDF files can be compressed to reduce their size, which can make them easier to share and download. However, compression can also affect the quality of the images and other content within the file.

The size of a PDF file can be used as an indicator of potential malware activity. Malware authors may use techniques such as obfuscation or encryption to hide the malicious code within the PDF

file. However, these techniques can increase the size of the PDF file. Therefore, a PDF file that is significantly larger than expected may be an indicator of malicious activity.

On the other hand, a PDF file that is significantly smaller than expected may also be an indicator of malicious activity. Malware authors may use techniques such as compression to reduce the size of the PDF file. However, compression can also be used to hide the malicious code within the file. Therefore, a PDF file that is smaller than expected may also be an indicator of malicious activity.

It is important to note that the size of a PDF file alone is not a definitive indicator of malware activity. A large or small PDF file may be legitimate, depending on the content and purpose of the document. However, analyzing the size of a PDF file can be a useful tool for detecting potential malware activity.

2.2.2 **Title Characters**

PDF files are commonly used for sharing and exchanging documents due to their flexibility and compatibility with almost all devices. However, like any other file format, PDF files can be used to deliver malware, making them a popular vector for cybercriminals. One method of detecting malware within a PDF file is by analyzing the title characters within the document. In this part, we will explore the use of PDF title characters for PDF malware detection.

PDF malware is a type of malware that is embedded within a PDF file. This malware can be designed to exploit vulnerabilities in PDF reader software, trick users into downloading and executing the malware, or to steal information, compromise systems, and launch further attacks.

Malware can be hidden within PDF files, disguised as legitimate documents or scripts. Such documents can also be used to download additional malware from the internet or infect the device on which they are opened.

The PDF title characters are the text that appears in the title bar of a PDF reader software when a PDF file is opened. This title is often used to provide users with information about the contents of the document, such as the name of the document, author, or publisher.

PDF files can have a title set within the document properties. The title can be added using a PDF authoring tool or software. The title can be set to any string of characters or a blank string. When a PDF file is opened in a PDF reader software, the title appears in the title bar of the reader.

The title of a PDF file can be used as an indicator of potential malware activity. Cybercriminals often use social engineering techniques to trick users into opening a malicious PDF file. They may use a catchy or intriguing title to lure the user into opening the document. For example, a title like "2019 tax return" may entice a user to open the PDF, which may contain malware.

PDF files that contain no title or have an empty title field can also be an indicator of potential malware activity. Cybercriminals may use these methods to try and hide the malicious nature of the PDF file.

Using title characters for malware detection is a quick and easy way to determine if a PDF file is suspicious. However, it is important to note that the title of a PDF file alone is not a definitive indicator of malware activity.

2.2.3 **Encryption**

PDF encryption is a security feature that is designed to protect PDF files from unauthorized access. Encryption involves scrambling the contents of a PDF file using a secret key, which can only be accessed by authorized users who have the correct decryption key. While encryption is primarily used for security purposes, it can also be a useful tool for detecting malware within PDF files. In this part, we will explore the use of PDF encryption for PDF malware detection.

PDF malware is a type of malware that is embedded within a PDF file. This malware can be designed to exploit vulnerabilities in PDF reader software, trick users into downloading and executing the malware, or to steal information, compromise systems, and launch further attacks.

PDF malware can be difficult to detect because it can be hidden within legitimate-looking PDF files. Cybercriminals may use social engineering techniques to trick users into opening a malicious PDF file, or they may use obfuscation techniques to hide the malware from detection.

PDF encryption is a security feature that is designed to protect PDF files from unauthorized access. Encryption involves scrambling the contents of a PDF file using a secret key, which can only be accessed by authorized users who have the correct decryption key. PDF encryption can

be used to protect sensitive information, such as financial data, medical records, and personal information.

PDF encryption can be applied to both the contents of the PDF file and the metadata, such as the title, author, and creation date. PDF encryption can be applied using a variety of encryption algorithms, including AES (Advanced Encryption Standard), RC4 (Rivest Cipher 4), and 3DES (Triple Data Encryption Standard).

PDF encryption can be a useful tool for detecting malware within PDF files. When a PDF file is encrypted, the contents of the file are scrambled using a secret key, which makes it difficult for cybercriminals to modify the contents of the file without the correct decryption key.

If a PDF file is encrypted, and it contains malware, the malware will also be encrypted. This means that when the PDF file is opened, the PDF reader software will not be able to access the malware until the file has been decrypted using the correct decryption key. If the decryption key is not available, the malware will remain encrypted and will not be able to execute.

By using PDF encryption, organizations can protect their sensitive information from unauthorized access and detect potential malware activity within their PDF files. PDF encryption can be used in conjunction with other security measures, such as antivirus software, firewalls, and intrusion detection systems, to provide a comprehensive security solution.

PDF encryption is a powerful security feature that is designed to protect PDF files from unauthorized access. Encryption can also be a useful tool for detecting potential malware activity within PDF files. If a PDF file is encrypted and it contains malware, the malware will also be encrypted, making it difficult for cybercriminals to modify the contents of the file without the correct decryption key.

While PDF encryption can be a useful tool for detecting malware, it should not be relied upon as the sole method of malware detection. Organizations should implement a comprehensive security solution that includes antivirus software, firewalls, intrusion detection systems, and user education to protect their systems from malware attacks.

2.2.4 **Metadata Size**

PDF metadata is a set of information that provides additional details about a PDF file, such as the file's title, author, creation date, and modification date. While metadata is primarily used to help users organize and manage their PDF files, it can also be a useful tool for detecting malware within PDF files. In this part, we will explore the use of PDF metadata size for PDF malware detection.[38]

PDF malware is a type of malware that is embedded within a PDF file. This malware can be designed to exploit vulnerabilities in PDF reader software, trick users into downloading and executing the malware, or to steal information, compromise systems, and launch further attacks.

PDF malware can be difficult to detect because it can be hidden within legitimate-looking PDF files. Cybercriminals may use social engineering techniques to trick users into opening a malicious PDF file, or they may use obfuscation techniques to hide the malware from detection.

PDF metadata is a set of information that provides additional details about a PDF file. Metadata can include information such as the file's title, author, creation date, modification date, and keywords. Metadata can be used to help users organize and manage their PDF files, and it can also be used by search engines to index and categorize PDF files.

Metadata is stored within the PDF file itself and can be accessed using PDF reader software or other tools that are designed to extract metadata from PDF files. Metadata can be modified using a variety of tools, including PDF authoring software, command-line tools, and scripts.

PDF metadata size can be a useful tool for detecting malware within PDF files. When a PDF file is created, metadata is added to the file to provide additional information about the file. The metadata size can be measured in bytes, and it can be used to determine whether a PDF file contains more information than is necessary.

If a PDF file contains a large number of metadata, it may indicate that the file has been modified or tampered with. Large amounts of metadata can also be an indication of malicious activity within the PDF file, such as the addition of malware code.

By measuring the metadata size of a PDF file, organizations can detect potential malware activity within their PDF files. If a PDF file contains a large number of metadata, it may be a sign that

the file has been tampered with or contains malware code. Organizations can use this information to take appropriate action to protect their systems from potential malware attacks.

PDF metadata size can be a useful tool for detecting potential malware activity within PDF files. If a PDF file contains a large number of metadata, it may be an indication that the file has been modified or tampered with, or that it contains malware code. By measuring the metadata size of a PDF file, organizations can detect potential malware activity and take appropriate action to protect their systems from malware attacks.

While PDF metadata size can be a useful tool for detecting malware, it should not be relied upon as the sole method of malware detection. [39] Organizations should implement a comprehensive security solution that includes antivirus software, firewalls, intrusion detection systems, and user education to protect their systems from malware attacks.

2.2.5 **Page Number**

PDF files are a common target for malware attacks due to their widespread use in sharing and distributing documents. Malware can be hidden in PDF files, which can infect the user's computer or steal sensitive information. In this part, we will explore the use of PDF page numbers for detecting malware in PDF files.

PDF malware is a type of malware that is embedded within a PDF file. This malware can be designed to exploit vulnerabilities in PDF reader software, trick users into downloading and executing the malware, or to steal information, compromise systems, and launch further attacks. Malware can be hidden in various parts of the PDF file, such as in embedded links, images, or scripts.

PDF page numbers are used to identify the page number of a specific page in a PDF file. Page numbers are usually located in the header or footer of a document, and they are useful for navigating and referencing specific pages in a long document.

PDF page numbers can be a useful tool for detecting malware in PDF files. Malware may be hidden on specific pages of a PDF file, such as on the last page, to avoid detection. By analyzing the page numbers of a PDF file, it may be possible to identify if the PDF file contains malware.

One way to use page numbers for malware detection is to analyze the distribution of page numbers in a PDF file. For example, a legitimate PDF file typically has a consistent and continuous page numbering system, where the page numbers increase sequentially from the beginning to the end of the document. In contrast, a PDF file that contains malware may have page numbers that are discontinuous or irregular.

Another way to use page numbers for malware detection is to analyze the page numbers of embedded links or scripts within the PDF file. Malware may be hidden in links or scripts that are embedded within a PDF file, and these links or scripts may be associated with a specific page number. By analyzing the page numbers of embedded links or scripts, it may be possible to identify if the PDF file contains malware.

PDF page numbers can be a useful tool for detecting malware in PDF files. By analyzing the distribution of page numbers and the page numbers of embedded links or scripts, it may be possible to identify if a PDF file contains malware. However, PDF page numbers should not be relied upon as the sole method of malware detection. Organizations should implement a comprehensive security solution that includes antivirus software, firewalls, intrusion detection systems, and user education to protect their systems from malware attacks.

2.2.6 **Header**

PDF files are a popular way of sharing and distributing digital documents. Unfortunately, PDF files are also a popular method for distributing malware. Malicious actors can hide malware in PDF files and use social engineering tactics to convince users to download and open them. In this part, we will discuss how the PDF header can be used for malware detection.

The PDF header is the first part of a PDF file and contains important information about the file. The header is usually the first 1024 bytes of the file and contains metadata that is used by PDF readers to interpret and display the file. The PDF header contains information such as the file version, the number of pages, and the encryption type.

Malicious PDF files may contain outdated PDF versions, which could be an indicator of malware. For example, older PDF versions may not have the same security features as newer versions, making them easier to exploit. PDF readers may also be less likely to detect malware in older versions of PDF files.

The PDF header contains metadata that can be used to identify potential indicators of malware. For example, a PDF file with an unusually large number of metadata fields or metadata fields with suspicious names may indicate that the file is malicious.

Malicious actors may encrypt PDF files to hide malware. However, legitimate PDF files can also be encrypted. One way to determine if a PDF file is encrypted is to analyze the PDF header. Encrypted PDF files will have a flag in the PDF header that indicates encryption is being used.

Malicious actors may use JavaScript to hide malware in PDF files. JavaScript can be used to create dynamic content and to interact with the user's computer. To detect JavaScript in PDF files, the PDF header can be analyzed for the presence of a /Names object. The /Names object is used to define named JavaScript actions, which may indicate the presence of malicious JavaScript.

The PDF header can be a useful tool for detecting malware in PDF files. By analyzing the PDF version, metadata, encryption, and JavaScript, it may be possible to identify potential indicators of malware. However, the PDF header should not be relied upon as the sole method of malware detection. Organizations should implement a comprehensive security solution that includes antivirus software, firewalls, intrusion detection systems, and user education to protect their systems from malware attacks.

2.2.7 **Image Number**

PDF files are commonly used for sharing and distributing digital documents. However, they can also be used for malicious purposes, such as distributing malware. Malware can be hidden within the images embedded in PDF files. In this part, we will discuss how PDF image numbers can be used for malware detection.

PDF image numbers are used to identify the position of an image within a PDF file. PDF files can contain multiple images, and each image is assigned a unique identifier, known as an object number. The object number can be used to determine the position of the image within the PDF file.

PDF image numbers can be a useful tool for detecting malware in PDF files. By analyzing the distribution of image numbers in a PDF file, it may be possible to identify if the file contains malware.

One way to use image numbers for malware detection is to analyze the distribution of image numbers in a PDF file. Legitimate PDF files typically have a continuous sequence of image numbers. In contrast, a PDF file that contains malware may have irregular or discontinuous image numbers. Malicious actors may use irregular image numbers to hide malware within the file.

Another way to use image numbers for malware detection is to analyze the object numbers of embedded images within the PDF file. Malware may be hidden in images that are embedded within a PDF file, and these images may be associated with a specific object number. By analyzing the object numbers of embedded images, it may be possible to identify if the PDF file contains malware.

It is important to note that image numbers should not be relied upon as the sole method of malware detection. Malicious actors may use various techniques to hide malware within a PDF file, and analyzing image numbers alone may not be sufficient to detect all types of malware.

PDF image numbers can be a useful tool for detecting malware in PDF files. By analyzing the distribution of image numbers and the object numbers of embedded images, it may be possible to identify if a PDF file contains malware. However, image numbers should not be relied upon as the sole method of malware detection. Organizations should implement a comprehensive security solution that includes antivirus software, firewalls, intrusion detection systems, and user education to protect their systems from malware attacks.

2.2.8 **Text**

PDF files are commonly used for sharing and distributing digital documents. However, they can also be used for malicious purposes, such as distributing malware. Malware can be hidden within the text of a PDF file. In this part, we will discuss how PDF text can be used for malware detection.

PDF files can contain different types of text, such as regular text, annotations, and form fields. Regular text is the text that is visible in the PDF file. Annotations are used to provide additional

information about the content in the PDF file. Form fields are used to collect information from the user.

PDF text can be a useful tool for detecting malware in PDF files. By analyzing the distribution of text within a PDF file, it may be possible to identify if the file contains malware.

One way to use PDF text for malware detection is to analyze the number of text objects in a PDF file. Legitimate PDF files typically have a low number of text objects, while PDF files that contain malware may have a large number of text objects. Malicious actors may use a high number of text objects to hide malware within the file.

Another way to use PDF text for malware detection is to analyze the content of the text within a PDF file. Malware may be hidden within the text of a PDF file, and this text may contain suspicious or malicious content. By analyzing the content of the text within a PDF file, it may be possible to identify if the file contains malware.

It is important to note that analyzing PDF text alone may not be sufficient to detect all types of malware. Malicious actors may use various techniques to hide malware within a PDF file, and analyzing text alone may not be enough to detect all types of malware.

PDF text can be a useful tool for detecting malware in PDF files. By analyzing the number of text objects and the content of the text within a PDF file, it may be possible to identify if the file contains malware. However, text analysis should not be relied upon as the sole method of malware detection.

2.2.9 **Object Number**

PDF is a widely used file format for documents and other types of content that need to be viewed or printed in a consistent manner regardless of the operating system or software used. However, PDF files can also be used to deliver malware or other malicious content to unsuspecting users. PDF object numbers are a key element in detecting PDF malware.

PDF files are made up of a series of objects, which are identified by a unique object number. These objects can be of different types, such as text, images, and annotations. Each object has a specific structure that includes a dictionary of properties and a stream of data.

PDF object numbers can be used to detect PDF malware in a number of ways. One approach is to analyze the structure of the PDF file and look for anomalies in the object numbering scheme. For example, some malware may use non-standard or random object numbers, which can indicate that the file is suspicious.

Another approach is to look for patterns in the object numbering scheme that are commonly used by malware. For example, some malware may use object numbers that are sequential or have a specific pattern, such as 1, 2, 3, 4, or 10, 20, 30, 40. By identifying these patterns, it may be possible to detect and prevent the spread of malware.

Anti-malware tools can use PDF object numbers to detect and prevent the spread of malware. These tools can analyze PDF files for anomalies in the object numbering scheme and other indicators of malware. For example, some anti-malware tools can detect PDF files that have been obfuscated or encrypted to hide their true contents.

In addition, anti-malware tools can use machine learning algorithms to analyze patterns in the object numbering scheme and other features of PDF files to identify new and emerging types of malware. These tools can also use heuristics to detect potentially malicious behavior, such as attempts to download additional files or execute commands on the user's system.

PDF object numbers are an important element in detecting PDF malware. By analyzing the structure of PDF files and looking for anomalies in the object numbering scheme, it may be possible to detect and prevent the spread of malware. Anti-malware tools can use PDF object numbers to identify new and emerging types of malware, and to detect potentially malicious behavior. As PDF files continue to be a popular means of delivering content, it is important to stay vigilant and use the latest security measures to protect against malware.

2.2.10 **Font Objects**

PDF files are commonly used for sharing and storing documents across various platforms. However, they can also be used to deliver malware to unsuspecting users. PDF font objects are a critical component in detecting PDF malware. In this part, we will explore how PDF font objects are used for malware detection.

PDF files can contain various types of objects, such as text, images, and fonts. Fonts are used to display text in PDF files and are usually embedded within the document itself. A font object in a PDF file includes information about the font type, size, and style.

PDF font objects can be used to detect malware in several ways. One approach is to analyze the font objects within a PDF file and look for anomalies in the font information. For example, malware can use non-standard or unusual fonts that are not commonly used in legitimate documents. By analyzing the font objects, it is possible to detect suspicious files.

Another approach is to look for patterns in the font objects that are commonly used by malware. For example, some malware may use a specific font type or size consistently across different PDF files. By identifying these patterns, it may be possible to detect and prevent the spread of malware.

Anti-malware tools can use PDF font objects to detect and prevent the spread of malware. These tools can analyze PDF files for anomalies in the font information and other indicators of malware. For example, some anti-malware tools can detect PDF files that contain encrypted or obfuscated font data.

In addition, anti-malware tools can use machine learning algorithms to analyze patterns in the font objects and other features of PDF files to identify new and emerging types of malware. These tools can also use heuristics to detect potentially malicious behavior, such as attempts to download additional files or execute commands on the user's system.

PDF font objects are an essential component in detecting PDF malware. By analyzing the font information within a PDF file, it is possible to detect suspicious files and prevent the spread of malware. Anti-malware tools can use PDF font objects to identify new and emerging types of malware, as well as to detect potentially malicious behavior. As PDF files continue to be a popular means of delivering content, it is crucial to stay vigilant and use the latest security measures to protect against malware.

2.2.11 **No. of Embedded Files**

PDF files are widely used for sharing documents and other types of content. However, PDF files can also be used to deliver malware to unsuspecting users. The number of embedded files in a

PDF file is a critical component in detecting PDF malware. In this part, we will explore how the number of embedded files in a PDF file is used for malware detection.

PDF files can contain embedded files such as images, videos, and other documents. These embedded files can be used to enhance the functionality and visual appeal of the PDF document. Embedded files are typically compressed within the PDF file and can be accessed by users when they interact with the PDF file.

The number of embedded files in a PDF file can be used to detect malware in several ways. One approach is to analyze the number of embedded files in a PDF file and look for unusual patterns. For example, malware may contain many embedded files or a small number of files that are unusually large. By analyzing the number of embedded files, it is possible to detect suspicious files.

Another approach is to look for patterns in the types of embedded files that are commonly used by malware. For example, some malware may use a specific type of embedded file or a specific file format consistently across different PDF files. By identifying these patterns, it may be possible to detect and prevent the spread of malware.

Anti-malware tools can use the number of embedded files in a PDF file to detect and prevent the spread of malware. These tools can analyze PDF files for anomalies in the number and type of embedded files and other indicators of malware. For example, some anti-malware tools can detect PDF files that contain encrypted or obfuscated embedded files.

In addition, anti-malware tools can use machine learning algorithms to analyze patterns in the number and type of embedded files and other features of PDF files to identify new and emerging types of malwares. These tools can also use heuristics to detect potentially malicious behavior, such as attempts to download additional files or execute commands on the user's system.

The number of embedded files in a PDF file is an essential component in detecting PDF malware. By analyzing the number and type of embedded files, it is possible to detect suspicious files and prevent the spread of malware. Anti-malware tools can use the number of embedded files to identify new and emerging types of malwares, as well as to detect potentially malicious behavior. As PDF files continue to be a popular means of delivering content, it is crucial to stay vigilant and use the latest security measures to protect against malware.

2.2.12 **Average Size of all the Embedded Data**

PDF files are commonly used for sharing documents and other types of content. However, PDF files can also be used to deliver malware to unsuspecting users. The average size of all the embedded files in a PDF file is a critical component in detecting PDF malware. In this part, we will explore how the average size of all the embedded files in a PDF file is used for malware detection.

PDF files can contain embedded files such as images, videos, and other documents. These embedded files can be used to enhance the functionality and visual appeal of the PDF document. Embedded files are typically compressed within the PDF file and can be accessed by users when they interact with the PDF file.

The average size of all the embedded files in a PDF file can be used to detect malware in several ways. One approach is to analyze the average size of all the embedded files in a PDF file and look for unusual patterns. For example, malware may contain a large number of embedded files that are unusually small or large. By analyzing the average size of all the embedded files, it is possible to detect suspicious files.

Another approach is to look for patterns in the types of embedded files that are commonly used by malware. For example, some malware may use a specific type of embedded file or a specific file format consistently across different PDF files. By identifying these patterns, it may be possible to detect and prevent the spread of malware.

Anti-malware tools can use the average size of all the embedded files in a PDF file to detect and prevent the spread of malware. These tools can analyze PDF files for anomalies in the average size and type of embedded files and other indicators of malware. For example, some anti-malware tools can detect PDF files that contain encrypted or obfuscated embedded files.

In addition, anti-malware tools can use machine learning algorithms to analyze patterns in the average size and type of embedded files and other features of PDF files to identify new and emerging types of malwares. These tools can also use heuristics to detect potentially malicious behavior, such as attempts to download additional files or execute commands on the user's system.

The average size of all the embedded files in a PDF file is an essential component in detecting PDF malware. By analyzing the average size and type of embedded files, it is possible to detect suspicious files and prevent the spread of malware. Anti-malware tools can use the average size of all the embedded files to identify new and emerging types of malwares, as well as to detect potentially malicious behavior. As PDF files continue to be a popular means of delivering content, it is crucial to stay vigilant and use the latest security measures to protect against malware.

2.3 Structural Features

Structural features are the features of a PDF that are directly related to the maliciousness of the malware. These include a variety of keywords and parameters. We have used 25 structural features here under 16 headings for simplicity purposes because some features are identical to other features. Details of these structural features are given below.

2.3.1 No. of Keywords “Streams”

PDF files are widely used for sharing documents, but they can also be used to distribute malware. To detect and prevent malware in PDF files, security researchers use a range of techniques, including analyzing the number of "streams" in a PDF file. In this part, we will explore the use of the number of "streams" as a technique for detecting PDF malware.

Streams are a fundamental component of PDF files, and they are used to store binary data, such as images, videos, and fonts, as well as other types of data, including JavaScript and metadata. Streams are usually compressed, which helps to reduce the size of the PDF file.

PDF files can have many streams, depending on the complexity of the document and the number of images, videos, and other types of data that are included in the file. Each stream in a PDF file has a unique identifier known as a stream ID.

The number of streams in a PDF file can be used to detect malware in several ways. One approach is to compare the number of streams in a PDF file to the expected number of streams for a file of similar complexity. If the number of streams in a PDF file is significantly higher or lower than the expected number, it may be an indication that the file contains malware.

Another approach is to look for patterns in the number of streams in PDF files that are known to contain malware. For example, malware may use a specific number of streams consistently across different PDF files. By identifying these patterns, it may be possible to detect and prevent the spread of malware.

In addition, some anti-malware tools can use heuristics to detect potentially malicious behavior based on the number of streams in a PDF file. For example, a tool may flag a PDF file as suspicious if it has an unusually large number of streams or if the number of streams increases significantly during the file's execution.

Malware authors may use multiple streams in a PDF file to evade detection by anti-malware tools. By splitting the malware into multiple streams, it can be more difficult for anti-malware tools to detect and remove the malware from the PDF file.

In addition, malware authors may use streams to hide malicious code within a PDF file. For example, malware may use JavaScript code embedded in a stream to execute commands on the user's system or to download additional malware.

The number of "streams" in a PDF file is an essential component in detecting PDF malware. By analyzing the number of streams and looking for unusual patterns, it is possible to detect suspicious files and prevent the spread of malware. Anti-malware tools can use the number of streams to identify new and emerging types of malware, as well as to detect potentially malicious behavior. As PDF files continue to be a popular means of delivering content, it is crucial to stay vigilant and use the latest security measures to protect against malware.

2.3.2 **No. of Keywords “endstreams”**

PDF files are widely used for sharing documents, but they can also be used to distribute malware. To detect and prevent malware in PDF files, security researchers use a range of techniques, including analyzing the number of "endstreams" keywords in a PDF file. In this part, we will explore the use of the number of "endstreams" keywords as a technique for detecting PDF malware.

In a PDF file, streams are typically compressed to reduce the size of the file. To correctly decompress a stream, a PDF reader must first locate the end of the stream. The end of a stream is indicated by the keyword "endstream" followed by a line break.

The number of "endstream" keywords in a PDF file indicates the number of compressed streams in the file. This information can be useful for detecting malware in PDF files.

The number of "endstream" keywords in a PDF file can be used to detect malware in several ways. One approach is to compare the number of "endstream" keywords in a PDF file to the expected number of "endstream" keywords for a file of similar complexity. If the number of "endstream" keywords in a PDF file is significantly higher or lower than the expected number, it may be an indication that the file contains malware.

Another approach is to look for patterns in the number of "endstream" keywords in PDF files that are known to contain malware. For example, malware may use a specific number of "endstream" keywords consistently across different PDF files. By identifying these patterns, it may be possible to detect and prevent the spread of malware.

In addition, some anti-malware tools can use heuristics to detect potentially malicious behavior based on the number of "endstream" keywords in a PDF file. For example, a tool may flag a PDF file as suspicious if it has an unusually large number of "endstream" keywords or if the number of "endstream" keywords increases significantly during the file's execution.

Malware authors may use multiple streams and "endstreams" keywords in a PDF file to evade detection by anti-malware tools. By splitting the malware into multiple streams, it can be more difficult for anti-malware tools to detect and remove the malware from the PDF file. Similarly, by using an unusual number of "endstream" keywords, malware authors can make it more difficult for anti-malware tools to detect and analyze the contents of a PDF file.

In addition, malware authors may use streams and "endstreams" keywords to hide malicious code within a PDF file. For example, malware may use JavaScript code embedded in a stream to execute commands on the user's system or to download additional malware.

The number of "endstreams" keywords in a PDF file is an essential component in detecting PDF malware. By analyzing the number of "endstreams" keywords and looking for unusual patterns, it is possible to detect suspicious files and prevent the spread of malware. Anti-malware tools can use the number of "endstreams" keywords to identify new and emerging types of malware, as well as to detect potentially malicious behavior. As PDF files continue to

be a popular means of delivering content, it is crucial to stay vigilant and use the latest security measures to protect against malware.

2.3.3 **Average Stream Size**

PDF files are widely used for sharing documents, but they can also be used to distribute malware. To detect and prevent malware in PDF files, security researchers use a range of techniques, including analyzing the average stream size in a PDF file. In this part, we will explore the use of the average stream size as a technique for detecting PDF malware.

Streams are a type of object in PDF files that contain compressed or uncompressed data, such as text or images. Streams are often used to reduce the size of a PDF file, as they can be compressed using algorithms like Flate, LZW, or RunLength. Streams are identified in a PDF file using the "stream" keyword and are ended using the "endstream" keyword.

The average stream size in a PDF file can be used to detect malware in several ways. One approach is to compare the average stream size in a PDF file to the expected average stream size for a file of similar complexity. If the average stream size in a PDF file is significantly higher or lower than the expected average, it may be an indication that the file contains malware.

Another approach is to look for patterns in the average stream size in PDF files that are known to contain malware. For example, malware may consistently use streams of a specific size across different PDF files. By identifying these patterns, it may be possible to detect and prevent the spread of malware.

In addition, some anti-malware tools can use heuristics to detect potentially malicious behavior based on the average stream size in a PDF file. For example, a tool may flag a PDF file as suspicious if it has an unusually large average stream size or if the average stream size increases significantly during the file's execution.

Malware authors may use different stream sizes in a PDF file to evade detection by anti-malware tools. By using varying stream sizes, it can be more difficult for anti-malware tools to detect and remove the malware from the PDF file. Similarly, by using streams of an unusual

size, malware authors can make it more difficult for anti-malware tools to detect and analyze the contents of a PDF file.

In addition, malware authors may use different stream sizes to hide malicious code within a PDF file. For example, malware may use JavaScript code embedded in a stream to execute commands on the user's system or to download additional malware.

The average stream size in a PDF file is an essential component in detecting PDF malware. By analyzing the average stream size and looking for unusual patterns, it is possible to detect suspicious files and prevent the spread of malware. Anti-malware tools can use the average stream size to identify new and emerging types of malware, as well as to detect potentially malicious behavior. As PDF files continue to be a popular means of delivering content, it is crucial to stay vigilant and use the latest security measures to protect against malware.

2.3.4 **No. of Xref Entries**

PDF files are a popular way of sharing documents. However, they can also be used to distribute malware. To detect and prevent malware in PDF files, security researchers use a range of techniques, including analyzing the number of Xref entries in a PDF file. In this part, we will explore the use of the number of Xref entries as a technique for detecting PDF malware.

Xref (cross-reference) entries are a fundamental component of a PDF file's internal structure. They provide information about the location of objects within the PDF file, including the pages, fonts, images, and other resources. The Xref table is a critical part of a PDF file and allows it to be rendered correctly.

The number of Xref entries in a PDF file can be used to detect malware in several ways. One approach is to compare the number of Xref entries in a PDF file to the expected number for a file of similar complexity. If the number of Xref entries in a PDF file is significantly higher or lower than the expected number, it may be an indication that the file contains malware.

Another approach is to look for patterns in the number of Xref entries in PDF files that are known to contain malware. For example, malware may consistently use a specific number of Xref entries across different PDF files. By identifying these patterns, it may be possible to detect and prevent the spread of malware.

In addition, some anti-malware tools can use heuristics to detect potentially malicious behavior based on the number of Xref entries in a PDF file. For example, a tool may flag a PDF file as suspicious if it has an unusually high number of Xref entries or if the number of Xref entries increases significantly during the file's execution.

Malware authors may use different numbers of Xref entries in a PDF file to evade detection by anti-malware tools. By using varying numbers of Xref entries, it can be more difficult for anti-malware tools to detect and remove the malware from the PDF file. Similarly, by using an unusual number of Xref entries, malware authors can make it more difficult for anti-malware tools to detect and analyze the contents of a PDF file.

In addition, malware authors may use different numbers of Xref entries to hide malicious code within a PDF file. For example, malware may use JavaScript code embedded in the Xref table to execute commands on the user's system or to download additional malware.

The number of Xref entries in a PDF file is an essential component in detecting PDF malware. By analyzing the number of Xref entries and looking for unusual patterns, it is possible to detect suspicious files and prevent the spread of malware. Anti-malware tools can use the number of Xref entries to identify new and emerging types of malware, as well as to detect potentially malicious behavior. As PDF files continue to be a popular means of delivering content, it is crucial to stay vigilant and use the latest security measures to protect against malware.

2.3.5 **No. of Name Obfuscations**

PDFs are widely used for sharing and storing digital documents. However, they can also be used to distribute malware. Malicious actors can embed malware into PDFs, which can then infect a computer system or network when the PDF is opened. In order to combat this threat, researchers and cybersecurity professionals have developed various methods for detecting PDF malware. One of these methods is the use of PDF number of name obfuscations.

PDF obfuscation is the practice of making code or data difficult to understand or analyze. This can be done in a variety of ways, including through the use of encryption, compression, and obfuscation techniques. One common technique is to obfuscate the names and numbers used in

the code. This can make it more difficult for malware detection tools to identify malicious code.

PDF number and name obfuscation is a technique used to obfuscate the names and numbers in a PDF document. This can be done in a variety of ways, such as by replacing the names and numbers with random characters or by using an encoding algorithm to transform the names and numbers into a different format. The goal of this technique is to make it more difficult for malware detection tools to identify the code in the PDF as malicious.

One way that PDF number and name obfuscation can be used for PDF malware detection is by analyzing the structure of the PDF document. Malware detection tools can examine the structure of the PDF and look for patterns that are indicative of obfuscation. For example, if the PDF contains a large number of randomly generated names or numbers, this may indicate that obfuscation is being used to conceal malicious code.

Another way that PDF number and name obfuscation can be used for PDF malware detection is by analyzing the code itself. Malware detection tools can deobfuscate the code and analyze it to determine whether it is malicious. This can be done by using reverse engineering techniques to analyze the obfuscated code and identify its purpose.

There are several tools and techniques that can be used to detect PDF number and name obfuscations. One commonly used tool is PDFid, which can be used to identify suspicious patterns in PDF documents. Other tools, such as PDF Stream Dumper and PDF Parser, can be used to extract and analyze the contents of a PDF document.

In conclusion, PDF number and name obfuscation is a technique used to obfuscate the names and numbers in a PDF document. This technique can be used to conceal malicious code and make it more difficult for malware detection tools to identify the code as malicious. However, there are several tools and techniques that can be used to detect PDF number and name obfuscations, making it possible to identify and neutralize PDF malware.

2.3.6 **Total Number of Filters**

PDFs are commonly used for sharing and storing digital documents. However, they can also be used to distribute malware. Malicious actors can embed malware into PDFs, which can then

infect a computer system or network when the PDF is opened. In order to combat this threat, researchers and cybersecurity professionals have developed various methods for detecting PDF malware. One of these methods is the use of PDF filters.

PDF filters are used to compress or decompress data within a PDF document. They can also be used to encode or decode data, or to transform data from one format to another. Malicious actors can use filters to hide malicious code within a PDF document. However, by analyzing the filters used in a PDF document, cybersecurity professionals can detect and neutralize PDF malware.

PDF filters are typically identified by their filter names, which are used to indicate the type of filter being used. Some examples of filter names include FlateDecode, ASCIIHexDecode, and RunLengthDecode. By analyzing the filter names used in a PDF document, cybersecurity professionals can identify suspicious patterns and detect potential malware.

One way that PDF filters can be used for PDF malware detection is by examining the filter order within a PDF document. Malicious actors may attempt to hide malicious code by using multiple filters in a specific order. By examining the filter order, cybersecurity professionals can detect suspicious patterns and identify potential malware.

Another way that PDF filters can be used for PDF malware detection is by analyzing the contents of the filters themselves. Malicious actors may attempt to use custom filters to hide malicious code within a PDF document. By analyzing the contents of the filters, cybersecurity professionals can detect suspicious patterns and identify potential malware.

There are several tools and techniques that can be used to detect PDF filters. One commonly used tool is PDFid, which can be used to identify suspicious patterns in PDF documents. Other tools, such as PDF Stream Dumper and PDF Parser, can be used to extract and analyze the contents of a PDF document.

In conclusion, PDF filters are commonly used to compress or decompress data within a PDF document. They can also be used to encode or decode data, or to transform data from one format to another. Malicious actors can use filters to hide malicious code within a PDF document. However, by analyzing the filters used in a PDF document, cybersecurity professionals can detect and neutralize PDF malware. There are several tools and techniques

that can be used to detect PDF filters, making it possible to identify and neutralize PDF malware.

2.3.7 **No. of Objects with Nested Filters**

PDFs are a widely used format for sharing digital documents, but they can also be used for malicious purposes. Cybercriminals can use PDFs to embed malware and infect computer systems or networks when the document is opened. To combat this threat, cybersecurity professionals and researchers have developed various techniques for detecting PDF malware, including analyzing the number of objects and nested filters within a PDF.

PDF objects are the building blocks of a PDF document. Each object can contain different types of data, including text, images, and metadata. Objects are typically identified by their object number and generation number. Objects in a PDF document can also be nested, meaning that one object can contain other objects within it.

PDF filters, as mentioned before, are used to compress, or decompress data within a PDF document. They can also be used to encode or decode data, or to transform data from one format to another. Malicious actors can use filters to hide malicious code within a PDF document, and one of the ways they can achieve this is by nesting filters within objects.

By analyzing the number of objects and nested filters within a PDF document, cybersecurity professionals can identify suspicious patterns and detect potential malware. Malicious actors may use a large number of objects and nested filters in an attempt to conceal malicious code. They may also use specific filter combinations or orders to bypass detection.

One technique for detecting PDF malware through object and filter analysis is to use a tool like PDFiD or PDF Stream Dumper. These tools can be used to extract the contents of a PDF document and analyze the number of objects and nested filters within it. By identifying suspicious patterns, these tools can alert cybersecurity professionals to potential malware.

Another technique for detecting PDF malware through object and filter analysis is to use a tool like PDF Parser. This tool can be used to extract and deobfuscate the contents of a PDF document. By analyzing the deobfuscated code, cybersecurity professionals can identify the

purpose of each object and filter within the document. This can help them to identify any suspicious or malicious code and take action to neutralize the threat.

In conclusion, PDFs can be used to distribute malware, and malicious actors can use nested filters within objects to conceal malicious code. By analyzing the number of objects and nested filters within a PDF document, cybersecurity professionals can detect potential malware and take action to neutralize the threat. There are several tools and techniques that can be used for this analysis, including PDFiD, PDF Stream Dumper, and PDF Parser. By using these tools and techniques, cybersecurity professionals can keep computer systems and networks safe from PDF malware.

2.3.8 **No. of Stream Object (ObjStm)**

PDFs are a popular file format used for sharing digital documents, but they can also be used to distribute malware. Malicious actors can embed malware within a PDF document using various techniques, including the use of stream objects (ObjStm). ObjStm is a PDF object type used to store multiple objects within a compressed stream.

By analyzing the number of ObjStm objects within a PDF document, cybersecurity professionals can detect potential malware and take action to neutralize the threat. Malicious actors may use ObjStm to hide malicious code within a PDF document, making it difficult to detect.

One way to detect PDF malware using ObjStm is to use a tool like PDFiD or PDF Stream Dumper. These tools can extract the contents of a PDF document and analyze the number of ObjStm objects within it. If the number of ObjStm objects is unusually high or if there are many nested ObjStm objects, it could be an indication of potential malware.

Another way to detect PDF malware using ObjStm is to examine the contents of the ObjStm object itself. Malicious actors may use ObjStm to store encrypted or obfuscated code, making it difficult to detect. However, by analyzing the contents of the ObjStm object, cybersecurity professionals can identify suspicious patterns and detect potential malware.

In addition to analyzing the number of ObjStm objects within a PDF document, cybersecurity professionals can also analyze the contents of the PDF document itself. By examining the

JavaScript code within the PDF, they can identify suspicious patterns and detect potential malware. JavaScript is a commonly used programming language within PDFs, and it can be used to execute malicious code.

Another way to detect PDF malware is by examining the metadata within the PDF document. Malicious actors may attempt to hide metadata that contains information about the PDF's origin or author. By examining the metadata, cybersecurity professionals can identify suspicious patterns and detect potential malware.

In conclusion, ObjStm is a PDF object type used to store multiple objects within a compressed stream, and it can be used to hide malicious code within a PDF document. By analyzing the number of ObjStm objects within a PDF document and examining the contents of those objects, cybersecurity professionals can detect potential malware and take action to neutralize the threat. There are several tools and techniques that can be used for this analysis, including PDFiD, PDF Stream Dumper, and examining the PDF's metadata and JavaScript code. By using these tools and techniques, cybersecurity professionals can keep computer systems and networks safe from PDF malware.

2.3.9 **No. of Keywords "/JS", No. of Keywords "/JavaScript"**

PDFs are a popular file format used for sharing digital documents, but they can also be used to distribute malware. Malicious actors can embed malware within a PDF document using various techniques, including the use of JavaScript code. JavaScript is a commonly used programming language within PDFs, and it can be used to execute malicious code [40]. Cybersecurity professionals can detect potential malware in a PDF document by analyzing the number of keywords "/JS" and "/JavaScript."[41]

The keyword "/JS" is used within a PDF document to indicate the presence of JavaScript code. When this keyword is detected, it may indicate the presence of malicious code. The keyword "/JavaScript" is also used within a PDF document to indicate the presence of JavaScript code. By analyzing the number of occurrences of these keywords within a PDF document, cybersecurity professionals can detect potential malware.

One way to detect PDF malware using these keywords is to use a tool like PDFiD or PDF Stream Dumper. These tools can extract the contents of a PDF document and analyze the

number of occurrences of "/JS" and "/JavaScript" keywords within it. If the number of occurrences of these keywords is unusually high or if they are located within an unusual object, it could be an indication of potential malware.

Figure 4: JavaScript in PDF raw

Another way to detect PDF malware using these keywords is to examine the contents of the JavaScript code itself. Malicious actors may use obfuscated or encrypted JavaScript code to conceal their malicious intentions. By examining the JavaScript code, cybersecurity professionals can identify suspicious patterns and detect potential malware.

In addition to analyzing the number of occurrences of these keywords within a PDF document, cybersecurity professionals can also analyze the metadata within the PDF document. By examining the metadata, they can identify suspicious patterns and detect potential malware.

Another way to detect PDF malware is to use a tool like PDF Parser. This tool can be used to extract and deobfuscate the contents of a PDF document. By analyzing the deobfuscated code, cybersecurity professionals can identify the purpose of each object and filter within the document. This can help them to identify any suspicious or malicious code and take action to neutralize the threat.

In conclusion, the keywords "/JS" and "/JavaScript" are used within a PDF document to indicate the presence of JavaScript code, which can be used to execute malicious code. By analyzing the number of occurrences of these keywords within a PDF document and examining the contents of the JavaScript code, cybersecurity professionals can detect potential malware. There are several tools and techniques that can be used for this analysis, including PDFiD, PDF Stream Dumper, PDF Parser, and examining the PDF's metadata. By using these tools and techniques, cybersecurity professionals can keep computer systems and networks safe from PDF malware.

2.3.10 **No. of Keywords "/URL", No. of Keywords "/Action"**

PDFs are a popular file format used for sharing digital documents. However, malicious actors can exploit PDFs to distribute malware by embedding malicious links or actions within the document. Cybersecurity professionals can detect potential malware in a PDF document by analyzing the number of occurrences of the keywords "/URL" and "/Action."

The keyword "/URL" is used within a PDF document to indicate the presence of a URL or hyperlink. When this keyword is detected, it may indicate the presence of a malicious link. The keyword "/Action" is used within a PDF document to indicate the presence of an action. Actions are events triggered by user interaction, such as clicking on a link or button. By analyzing the number of occurrences of these keywords within a PDF document, cybersecurity professionals can detect potential malware.

One way to detect PDF malware using these keywords is to use a tool like PDFiD or PDF Stream Dumper. These tools can extract the contents of a PDF document and analyze the number of occurrences of "/URL" and "/Action" keywords within it. If the number of occurrences of these keywords is unusually high or if they are located within an unusual object, it could be an indication of potential malware.

Another way to detect PDF malware using these keywords is to examine the contents of the URLs or actions themselves. Malicious actors may use obfuscated or encrypted URLs or actions to conceal their malicious intentions. By examining the URLs or actions, cybersecurity professionals can identify suspicious patterns and detect potential malware.

In addition to analyzing the number of occurrences of these keywords within a PDF document, cybersecurity professionals can also analyze the metadata within the PDF document. By examining the metadata, they can identify suspicious patterns and detect potential malware.

Another way to detect PDF malware is to use a tool like PDF Parser. This tool can be used to extract and deobfuscate the contents of a PDF document. By analyzing the deobfuscated code, cybersecurity professionals can identify the purpose of each object and filter within the document. This can help them to identify any suspicious or malicious URLs or actions and take action to neutralize the threat.

In conclusion, the keywords "/URL" and "/Action" are used within a PDF document to indicate the presence of a URL or hyperlink and an action, respectively. By analyzing the number of occurrences of these keywords within a PDF document and examining the contents of the URLs or actions, cybersecurity professionals can detect potential malware. There are several tools and techniques that can be used for this analysis, including PDFiD, PDF Stream Dumper, PDF Parser, and examining the PDF's metadata. By using these tools and techniques, cybersecurity professionals can keep computer systems and networks safe from PDF malware.

2.3.11 **No. of Keywords "/AA", No. of Keywords "/OpenAction"**

PDF files have become an increasingly popular file format for sharing documents online. Unfortunately, like any other file format, PDF files can be used to distribute malware. Malicious actors can exploit vulnerabilities in PDF software to execute code or launch phishing attacks, among other things. To help detect malicious PDF files, there are certain keywords that security researchers can look for within the file. Two of these keywords are "/AA" and "/OpenAction". In this part, we'll explore how these keywords can be used for PDF malware detection.

The "/AA" keyword in PDF refers to "additional actions". Additional actions are JavaScript scripts that are executed when a user interacts with a PDF document. For example, an additional action could be executed when the user clicks on a button or types of text into a field. Additional actions can be used for legitimate purposes, such as form validation, but they can also be used for malicious purposes, such as launching a malware download.

To detect whether a PDF file contains malicious additional actions, security researchers can use a PDF analysis tool that looks for the "/AA" keyword. The tool can then extract the JavaScript code associated with the additional action and analyze it for any malicious behavior. The analysis can include checking for the presence of known malware signatures or looking for suspicious network connections.

The "/OpenAction" keyword in PDF refers to an action that is executed when a PDF document is opened. Like additional actions, open actions can be used for legitimate purposes, such as opening the document to a specific page, but they can also be used for malicious purposes, such as launching a malware download.

To detect whether a PDF file contains a malicious open action, security researchers can again use a PDF analysis tool that looks for the "/OpenAction" keyword. The tool can then extract the action and analyze it for any malicious behavior. This can include checking for the presence of known malware signatures or looking for suspicious network connections.

While keyword-based detection can be effective for detecting some types of PDF malware, it has limitations. For example, some malware may use obfuscated or encrypted code that is designed to evade detection by analysis tools. In addition, not all malicious PDF files will contain the "/AA" or "/OpenAction" keywords, so a lack of these keywords does not necessarily mean that a file is safe.

PDF files can be used to distribute malware, but the /AA and /OpenAction keywords can be used to help detect malicious behavior. By using PDF analysis tools that look for these keywords, security researchers can extract and analyze JavaScript code associated with additional actions and open actions. While keyword-based detection has its limitations, it is an important tool in the fight against PDF malware. As always, it is important to keep your software up-to-date and exercise caution when opening PDF files from unknown sources.

2.3.12 **No. of Keywords “/launch”, No. of Keywords “/submitForm”**

PDF files are a common file format for sharing and distributing documents online. However, they can also be used to distribute malware. Malicious actors can exploit vulnerabilities in PDF software to execute code or launch phishing attacks, among other things. To help detect malicious PDF files, there are certain keywords that security researchers can look for within the

file. Two of these keywords are "/launch" and "/submitForm". In this part, we'll explore how these keywords can be used for PDF malware detection.

The "/launch" keyword in PDF refers to an action that launches an application or executable file when the PDF is opened. This can be used for legitimate purposes, such as opening a linked document or a website, but it can also be used for malicious purposes, such as launching malware.

To detect whether a PDF file contains a malicious launch action, security researchers can use a PDF analysis tool that looks for the "/launch" keyword. The tool can then extract the action and analyze it for any malicious behavior. This can include checking for the presence of known malware signatures or looking for suspicious network connections.

The "/submitForm" keyword in PDF refers to an action that submits data from a PDF form to a server or other external location. This can be used for legitimate purposes, such as submitting a job application or survey, but it can also be used for malicious purposes, such as submitting sensitive data to a phishing site.

To detect whether a PDF file contains a malicious submitForm action, security researchers can use a PDF analysis tool that looks for the "/submitForm" keyword. The tool can then extract the action and analyze it for any malicious behavior. This can include checking for the presence of known malware signatures or looking for suspicious network connections.

While keyword-based detection can be effective for detecting some types of PDF malware, it has limitations. For example, some malware may use obfuscated or encrypted code that is designed to evade detection by analysis tools. In addition, not all malicious PDF files will contain the "/launch" or "/submitForm" keywords, so a lack of these keywords does not necessarily mean that a file is safe.

PDF files can be used to distribute malware, but the /launch and /submitForm keywords can be used to help detect malicious behavior. By using PDF analysis tools that look for these keywords, security researchers can extract and analyze actions associated with launching applications or submitting form data. While keyword-based detection has its limitations, it is an important tool in the fight against PDF malware. As always, it is important to keep your software up-to-date and exercise caution when opening PDF files from unknown sources.

2.3.13 **No. of Keywords “/Acroform”, No. of Keywords “/XFA”**

PDF files have become a popular file format for sharing and distributing documents online. Unfortunately, PDF files can also be used to distribute malware. Malicious actors can exploit vulnerabilities in PDF software to execute code or launch phishing attacks, among other things. To help detect malicious PDF files, there are certain keywords that security researchers can look for within the file. Two of these keywords are "/Acroform" and "/XFA". In this part, we'll explore how these keywords can be used for PDF malware detection.

The "/Acroform" keyword in PDF refers to a type of interactive form in a PDF file. Acroforms can be used for legitimate purposes, such as filling out a tax form, but they can also be used for malicious purposes, such as collecting sensitive information or launching malware.

To detect whether a PDF file contains a malicious Acroform, security researchers can use a PDF analysis tool that looks for the "/Acroform" keyword. The tool can then extract the form and analyze it for any malicious behavior. This can include checking for the presence of known malware signatures or looking for suspicious network connections.

The "/XFA" keyword in PDF refers to XML Forms Architecture, which is a way of creating interactive forms in a PDF file. XFA forms can be used for legitimate purposes, such as submitting an online job application, but they can also be used for malicious purposes, such as launching malware.

To detect whether a PDF file contains a malicious XFA form, security researchers can use a PDF analysis tool that looks for the "/XFA" keyword. The tool can then extract the form and analyze it for any malicious behavior. This can include checking for the presence of known malware signatures or looking for suspicious network connections.

While keyword-based detection can be effective for detecting some types of PDF malware, it has limitations. For example, some malware may use obfuscated or encrypted code that is

designed to evade detection by analysis tools. In addition, not all malicious PDF files will contain the "/Acroform" or "/XFA" keywords, so a lack of these keywords does not necessarily mean that a file is safe.

PDF files can be used to distribute malware, but the /Acroform and /XFA keywords can be used to help detect malicious behavior. By using PDF analysis tools that look for these keywords, security researchers can extract and analyze interactive forms associated with collecting data or launching applications. While keyword-based detection has its limitations, it is an important tool in the fight against PDF malware. As always, it is important to keep your software up-to-date and exercise caution when opening PDF files from unknown sources.

2.3.14 **No. of Keywords "/JBig2Decode", No. of Keywords "/Colors"**

PDF files have become a popular file format for sharing and distributing documents online. Unfortunately, PDF files can also be used to distribute malware. Malicious actors can exploit vulnerabilities in PDF software to execute code or launch phishing attacks, among other things. To help detect malicious PDF files, there are certain keywords that security researchers can look for within the file. Two of these keywords are "/JBig2Decode" and "/Colors". In this part, we'll explore how these keywords can be used for PDF malware detection.

The "/JBig2Decode" keyword in PDF refers to a compression method used for image data in PDF files. While JBIG2 is a legitimate compression method, it can also be used by malware to hide malicious code. Malicious actors can use JBIG2 to compress code in a PDF file, making it more difficult to detect.

To detect whether a PDF file contains malicious JBIG2 code, security researchers can use a PDF analysis tool that looks for the "/JBig2Decode" keyword. The tool can then extract the compressed image data and analyze it for any malicious behavior. This can include checking for the presence of known malware signatures or looking for suspicious network connections.

The "/Colors" keyword in PDF refers to the number of colors used in an image in a PDF file. While this may seem like a benign keyword, it can be used by malware to evade detection. Malicious actors can create PDF files with images that have a large number of colors, which can cause some analysis tools to fail or crash.

To detect whether a PDF file contains malicious code that exploits the "/Colors" keyword, security researchers can use a PDF analysis tool that looks for large images with a high number of colors. The tool can then analyze the image for any malicious behavior, such as hidden code or network connections.

While keyword-based detection can be effective for detecting some types of PDF malware, it has limitations. For example, some malware may use obfuscated or encrypted code that is designed to evade detection by analysis tools. In addition, not all malicious PDF files will contain the "/JBig2Decode" or "/Colors" keywords, so a lack of these keywords does not necessarily mean that a file is safe.

PDF files can be used to distribute malware, but the /JBig2Decode and /Colors keywords can be used to help detect malicious behavior. By using PDF analysis tools that look for these keywords, security researchers can extract and analyze compressed image data associated with PDF files. While keyword-based detection has its limitations, it is an important tool in the fight against PDF malware. As always, it is important to keep your software up-to-date and exercise caution when opening PDF files from unknown sources.

2.3.15 **No. of Keywords “/Richmedia”, No. of Keywords “/Trailer”**

PDF files are a popular document format that are widely used for sharing information. However, due to their flexibility, they can also be used for malicious purposes, such as spreading malware. To combat this, security researchers use various techniques to detect malware hidden within PDF files. Two of these techniques include looking for the presence of

the "/Richmedia" and "/Trailer" keywords within the PDF file. In this part, we'll explore how these keywords can be used for PDF malware detection.

The "/Richmedia" keyword is used in PDF files to embed multimedia content, such as video or audio, within a PDF document. However, this keyword can also be exploited by attackers to embed malicious code within the PDF file. This can be done by embedding a Flash file that contains malicious code, which is then executed when the PDF file is opened.

To detect whether a PDF file contains malicious code embedded within a "/Richmedia" object, security researchers can use a PDF analysis tool that searches for the keyword. Once the object is identified, the tool can then analyze the object for any malicious code, such as an embedded Flash file.

The "/Trailer" keyword is used in PDF files to provide information about the structure of the document, including its catalog and metadata. This keyword can also be used by attackers to hide malicious code within the PDF file. This is done by manipulating the "/Trailer" keyword to include additional information, which can be used to launch an attack.

To detect whether a PDF file contains malicious code hidden within the "/Trailer" keyword, security researchers can use a PDF analysis tool that searches for the keyword. The tool can then analyze the metadata within the "/Trailer" keyword for any suspicious activity, such as the presence of hidden scripts or URLs that may be used to launch an attack.

While keyword-based detection can be effective for detecting some types of PDF malware, it has limitations. Malware authors can use various techniques to obfuscate or hide their code, making it difficult to detect using simple keyword searches. In addition, not all malicious PDF files will contain the "/Richmedia" or "/Trailer" keywords, so a lack of these keywords does not necessarily mean that a file is safe.

PDF files can be used to distribute malware, but the "/Richmedia" and "/Trailer" keywords can be used to help detect malicious behavior. By using PDF analysis tools that look for these keywords, security researchers can identify embedded multimedia content and analyze the metadata associated with PDF files. While keyword-based detection has its limitations, it is an important tool in the fight against PDF malware. As always, it is important to keep your software up-to-date and exercise caution when opening PDF files from unknown sources.

2.3.16 **No. of Keywords “Xref”, No. of Keywords “/Startxref”**

PDF files are a popular document format that is commonly used for sharing information. However, malicious actors can use PDF files to spread malware, which makes it essential for security researchers to develop techniques to detect malware hidden within PDF files. Two of these techniques include looking for the presence of the "/Xref" and "/Startxref" keywords within the PDF file. In this part, we'll explore how these keywords can be used for PDF malware detection.

The "/Xref" keyword is used in PDF files to define the cross-reference table, which contains information about the location of objects within the PDF file. This keyword is essential for the PDF file to function properly, but it can also be used by attackers to embed malicious code within the PDF file. This is done by manipulating the cross-reference table to include additional information, which can be used to launch an attack.

To detect whether a PDF file contains malicious code hidden within the "/Xref" keyword, security researchers can use a PDF analysis tool that searches for the keyword. The tool can then analyze the cross-reference table for any suspicious activity, such as the presence of hidden scripts or URLs that may be used to launch an attack.

The "/Startxref" keyword is used in PDF files to provide information about the start of the cross-reference table. This keyword can also be used by attackers to hide malicious code within

the PDF file. This is done by manipulating the "/Startxref" keyword to include additional information, which can be used to launch an attack.

To detect whether a PDF file contains malicious code hidden within the "/Startxref" keyword, security researchers can use a PDF analysis tool that searches for the keyword. The tool can then analyze the metadata associated with the "/Startxref" keyword for any suspicious activity, such as the presence of hidden scripts or URLs that may be used to launch an attack.

While keyword-based detection can be effective for detecting some types of PDF malware, it has limitations. Malware authors can use various techniques to obfuscate or hide their code, making it difficult to detect using simple keyword searches. In addition, not all malicious PDF files will contain the "/Xref" or "/Startxref" keywords, so a lack of these keywords does not necessarily mean that a file is safe.

PDF files can be used to distribute malware, but the "/Xref" and "/Startxref" keywords can be used to help detect malicious behavior. By using PDF analysis tools that look for these keywords, security researchers can identify hidden scripts or URLs that may be used to launch an attack. While keyword-based detection has its limitations, it is an important tool in the fight against PDF malware. As always, it is important to keep your software up-to-date and exercise caution when opening PDF files from unknown sources.

Chapter 3

Architecture and Training of MLP

3.1 Overview

The Multi-Layer Perceptron is a type of artificial neural network that consists of multiple layers of interconnected nodes. Each node performs a simple mathematical operation on the input it receives and passes the result on to the next layer. The output of the final layer is the predicted class of the input data. MLPs have been successfully applied in a wide range of applications, including image classification, speech recognition, and natural language processing.

The architecture of an MLP consists of an input layer, one or more hidden layers, and an output layer. The input layer receives the input data, which is typically represented as a vector of features. Each node in the input layer corresponds to a feature of the input data. The hidden layers perform a series of transformations on the input data, gradually learning to extract higher-level features. Finally, the output layer produces the predicted class of the input data, which is typically represented as a probability distribution over the possible classes.[42]

The training of an MLP involves adjusting the weights of the connections between its nodes based on the training data. The weights control the strength of the connections between the nodes and determine how the input data is transformed as it passes through the network. The training process involves an iterative optimization algorithm called backpropagation, which adjusts the weights to minimize the difference between the predicted output and the actual output.

Backpropagation involves computing the gradient of the loss function with respect to the weights and using this gradient to update the weights. The loss function measures how well the MLP is able to predict the correct output given the input data. There are various loss functions that can be used for different types of classification problems, such as cross-entropy loss for binary classification or categorical cross-entropy loss for multi-class classification.

The optimization of MLPs involves finding the optimal values for the hyperparameters of the network, such as the learning rate, the number of hidden layers, and the number of nodes in each layer. Hyperparameter tuning can be a challenging task, as the optimal values may depend on the

specific characteristics of the dataset and the problem being solved. In the next paragraph, we will discuss the steps involved in optimizing MLPs for detecting PDF malware.

The optimization of MLPs for detecting PDF malware involves several steps, including dataset curation, feature selection, hyperparameter tuning, and model evaluation. Each of these steps is critical to the overall performance of the MLP and requires careful consideration.

Dataset curation involves selecting a set of PDF files that represent the target population of malware. The dataset should include a representative sample of benign and malicious PDF files, and the malicious files should be diverse in terms of the techniques used to hide the malware. The dataset should also be balanced to ensure that the MLP does not become biased towards the majority class.

Feature selection involves selecting a subset of the available features that are most relevant to the task of detecting PDF malware. This step can be challenging, as there may be many features to choose from, and the optimal subset may depend on the specific characteristics of the dataset and the problem being solved. Feature selection can be performed using techniques such as mutual information, correlation analysis, or principal component analysis (PCA).

Hyperparameter tuning involves finding the optimal values for the hyperparameters of the MLP, such as the learning rate, the number of hidden layers, and the number of nodes in each layer. Hyperparameter tuning can be performed using techniques such as grid search, random search, or Bayesian optimization. The optimal values for the hyperparameters can be determined using metrics such as accuracy, precision, recall, or F1-score.

Model evaluation involves testing the performance of the optimized MLP on a separate test set of PDF files that were not used in the training or validation phases. Model evaluation can be performed using metrics such as accuracy, precision, recall, or F1-score. The performance of the optimized MLP should be compared to that of other state-of-the-art methods for detecting PDF malware.

3.2 Experimental Setup

For the purpose of this thesis, we have decided to go with the dataset provided by the Canadian institute of cyber security with the name CIC-Evasive-PDFMal2022. This dataset has been created by collecting 11,173 malicious files from Contagio, 20,000 malicious files from Virustotal and 9,109 benign files from Contagio.[43]

The multi-layer perceptron model that we have used uses 3 hidden layers with 6,100,50 and 10 neurons respectively. Maximum number of iterations to be run by the solver are set to 5,000,000. 80% of the data was used for training and 20% for testing. After doing the experiment, we were able to achieve an accuracy of 96.05985. This is a very good accuracy which is very much in line with the best models available in the literature.

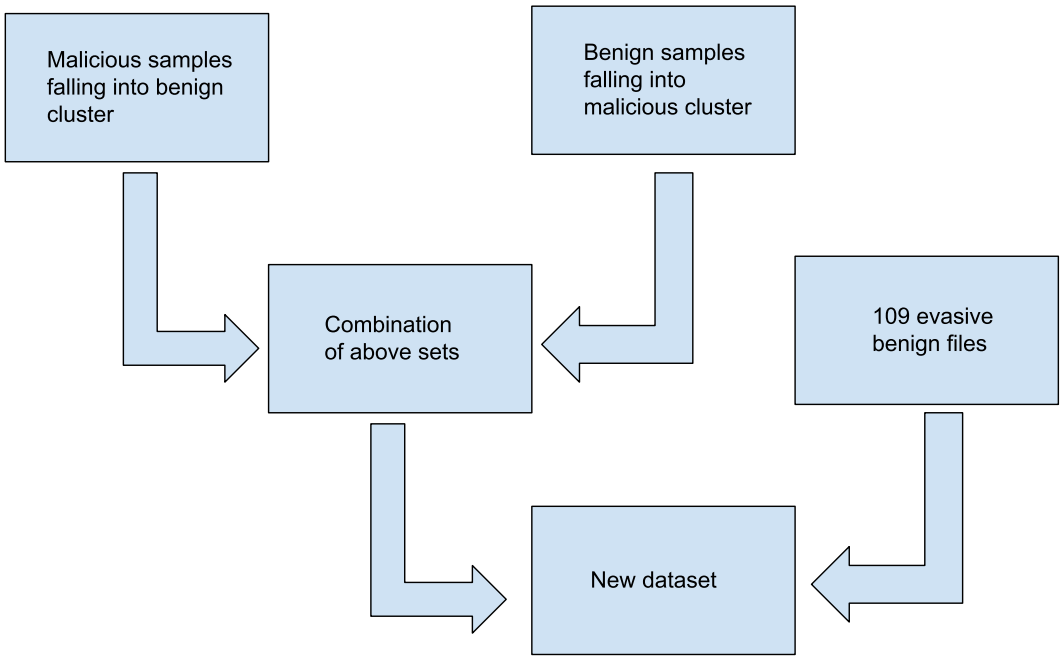

Figure 5: Compilation of Dataset

Next, we have proposed a comparison of MLP with other Artificial Intelligence algorithms and proved that MLP is the better algorithm for implementation of PDF malware detector using Artificial Intelligence

3.3 **Comparison with other models**

After performing the experiment with MLP, we compared our results with other algorithms as well. These algorithms include:

1. ExtraTreesClassifier
2. XGBClassifier
3. RandomForestClassifier
4. BaggingClassifier
5. AdaBoostClassifier
6. LGBMClassifier
7. DecisionTreeClassifier
8. KNeighborsClassifier
9. LabelPropagation
10. LabelSpreading
11. ExtraTreeClassifier
12. SVC
13. SGDClassifier
14. LogisticRegression
15. LinearDiscriminantAnalysis
16. RidgeClassifierCV
17. RidgeClassifier
18. CalibratedClassifierCV
19. LinearSVC
20. NuSVC
21. NearestCentroid
22. BernoulliNB
23. Perceptron
24. PassiveAggressiveClassifier
25. QuadraticDiscriminantAnalysis
26. GaussainNB
27. DummyClassifier

Comparison was made on the following metrics:

3.3.1 **Accuracy**

Accuracy is a commonly used evaluation metric in artificial intelligence (AI) and machine learning (ML) algorithms that measures how well a model is able to correctly predict the outcome of a task.

In the context of classification tasks, accuracy measures the proportion of correctly classified instances over the total number of instances in the dataset. It is calculated as:

accuracy = (number of correctly classified instances) / (total number of instances)

For example, if a model correctly classifies 90 out of 100 instances, the accuracy would be 90%.

While accuracy is a useful metric, it may not always be the best measure of performance in certain situations, especially when the class distribution is imbalanced or when different types of errors have different consequences. In these cases, other metrics such as precision, recall, F1 score, or AUC may be more appropriate.

3.3.2 **Balanced Accuracy**

Balanced accuracy is a modified version of accuracy that takes into account class imbalance in a dataset. It is a useful evaluation metric in binary classification tasks where the number of samples in each class is not equal.

In contrast to accuracy, which only considers the overall accuracy of the model, balanced accuracy calculates the average of sensitivity (true positive rate) and specificity (true negative rate) for each class. Sensitivity measures the proportion of true positives (TP) out of all positive cases (TP + false negatives (FN)), while specificity measures the proportion of true negatives (TN) out of all negative cases (TN + false positives (FP)).

The formula for balanced accuracy is:

balanced accuracy = (sensitivity + specificity) / 2

Balanced accuracy is especially useful when the class distribution is imbalanced, where one class has significantly more samples than the other. In such cases, accuracy may be a misleading metric because the model may achieve high accuracy by simply predicting the majority class. In contrast, balanced accuracy takes into account the performance of the model in both classes and provides a more accurate assessment of its performance.

3.3.3 **ROC AUC**

ROC (Receiver Operating Characteristic) curve is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is a plot of the true positive rate (TPR) against the false positive rate (FPR) for different threshold values.

AUC (Area Under the ROC Curve) is a metric that measures the overall performance of a binary classifier system. It is the area under the ROC curve, and its value ranges from 0 to 1. AUC is a useful metric because it summarizes the ROC curve into a single number, indicating how well the classifier is able to distinguish between positive and negative classes.

An AUC of 1.0 indicates perfect classification, while an AUC of 0.5 indicates that the classifier is no better than random guessing. AUC values between 0.5 and 1.0 indicate varying degrees of classification performance, with higher values indicating better performance.

In general, ROC and AUC are commonly used to evaluate the performance of binary classification models, especially in situations where the class distribution is imbalanced, i.e., one class has many more examples than the other.

3.3.4 **F1 Score**

The F1 score is a commonly used evaluation metric in artificial intelligence (AI) and machine learning (ML) algorithms to measure the overall performance of a model in binary classification tasks. It is the harmonic mean of precision and recall, and it takes into account both false positives and false negatives.

Precision measures the proportion of true positives (TP) out of all predicted positives (TP + false positives (FP)), while recall measures the proportion of true positives (TP) out of all actual positives (TP + false negatives (FN)). The F1 score is the harmonic mean of precision and recall, and it ranges from 0 to 1, with higher values indicating better performance.

The formula for F1 score is:

F1 score = 2 * ((precision * recall) / (precision + recall))

The F1 score is a useful metric when the dataset is imbalanced, i.e., one class has many more examples than the other. In such cases, accuracy may not be a good metric to evaluate the performance of the model, as it can be misleading. The F1 score provides a balanced measure of precision and recall, and it can help to identify models that have high precision but low recall, or vice versa.

3.3.5 **Time Taken**

The time taken in artificial intelligence algorithm comparison refers to the amount of time required by different algorithms to train a model, make predictions on a dataset, and evaluate its performance. It is an important factor to consider when comparing different algorithms because it can have a significant impact on the practical feasibility of using a model in a real-world setting.

The time taken for an algorithm to run depends on several factors, such as the size of the dataset, the complexity of the model, and the computing resources available. Some algorithms, such as decision trees and k-nearest neighbors, are relatively fast to train and make predictions, while others, such as deep neural networks, can be very computationally expensive.

When comparing different algorithms, it is important to consider the time taken in addition to other factors such as accuracy, interpretability, and scalability. Depending on the application, a fast but less accurate algorithm may be preferable over a slower but more accurate one. Therefore, the choice of algorithm ultimately depends on the specific requirements of the task at hand and the trade-offs between different factors.

3.4 **Results**

The results of the comparison based on above-mentioned evaluation metrics on the above-mentioned algorithms can be observed in the table below:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
ExtraTreesClassifier	1.00	1.00	1.00	1.00	0.32
XGBClassifier	1.00	1.00	1.00	1.00	0.20
RandomForestClassifier	1.00	1.00	1.00	1.00	0.44
BaggingClassifier	1.00	1.00	1.00	1.00	0.08
AdaBoostClassifier	1.00	1.00	1.00	1.00	0.40
LGBMClassifier	1.00	1.00	1.00	1.00	0.21
DecisionTreeClassifier	1.00	1.00	1.00	1.00	0.02
KNeighborsClassifier	0.99	0.99	0.99	0.99	0.16
LabelPropagation	0.99	0.99	0.99	0.99	1.12
LabelSpreading	0.99	0.99	0.99	0.99	2.37
ExtraTreeClassifier	0.99	0.99	0.99	0.99	0.02
SVC	0.99	0.99	0.99	0.99	0.49
SGDClassifier	0.98	0.98	0.98	0.98	0.04
LogisticRegression	0.97	0.97	0.97	0.97	0.04
LinearDiscriminantAnalysis	0.97	0.97	0.97	0.97	0.04
RidgeClassifierCV	0.97	0.97	0.97	0.97	0.02
RidgeClassifier	0.97	0.97	0.97	0.97	0.02
CalibratedClassifierCV	0.97	0.97	0.97	0.97	1.00

LinearSVC	0.96	0.96	0.96	0.96	0.25
NuSVC	0.95	0.96	0.96	0.95	2.89
NearestCentroid	0.95	0.95	0.95	0.95	0.02
BernoulliNB	0.94	0.94	0.94	0.94	0.02
MLP	0.88	0.88	0.88	0.88	0.02
PassiveAggressiveClassifier	0.88	0.89	0.89	0.88	0.02
QuadraticDiscriminantAnalysis	0.82	0.83	0.83	0.81	0.02
GaussianNB	0.80	0.82	0.82	0.80	0.02
DummyClassifier	0.56	0.50	0.50	0.40	0.02

Table 1: Comparison of MLP with other algorithms

There are other algorithms that perform better then MLP according to all these evaluation parameters except time taken. Below figure shows comparison of MLP with Random Forest and Decision Tree algorithms which are most popular in literature.

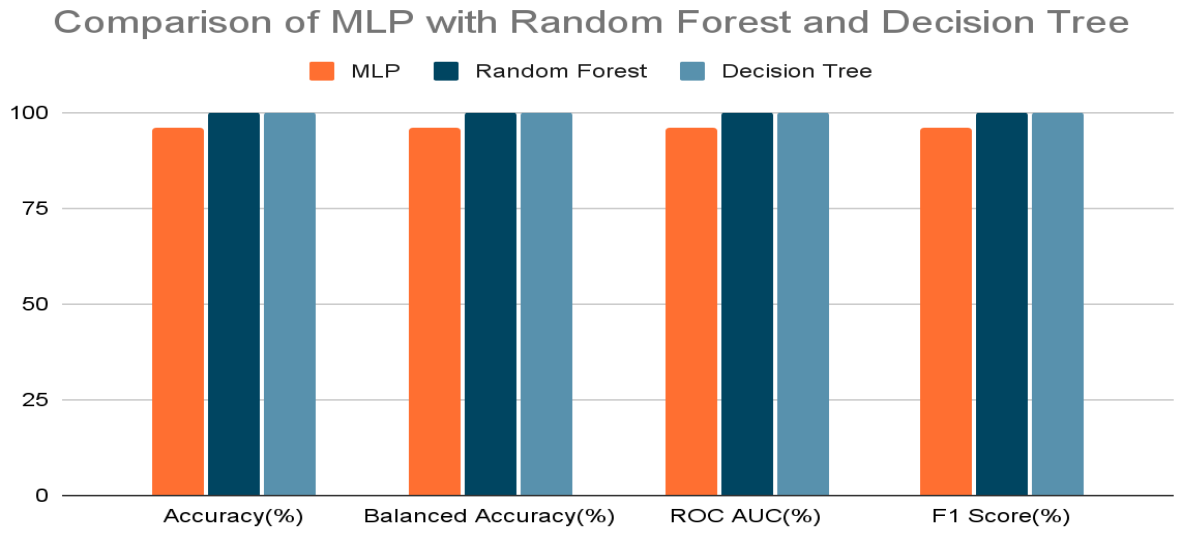

Figure 6: Comparison of MLP with Random Forest and Decision Tree

3.5 **Benefits of MLP**

Multi-layer perceptron (MLP) is a type of artificial neural network that can be used for classification and regression tasks, while decision trees and random forests are machine learning algorithms used for classification and regression tasks as well. Here are some potential benefits of using MLP over decision tree and random forest:

- MLP can handle non-linearly separable data: MLP can learn complex non-linear decision boundaries, whereas decision trees and random forests may struggle with this task.
- MLP can generalize better: MLP can generalize better to new, unseen data compared to decision trees and random forests, which may overfit the training data.
- MLP can work with continuous and categorical data: MLP can handle both continuous and categorical data, whereas decision trees and random forests typically work best with categorical data.
- MLP can learn complex feature representations: MLP can learn complex feature representations of the data, whereas decision trees and random forests may rely on hand-crafted feature engineering.
- MLP can be used for end-to-end learning: MLP can be used for end-to-end learning, where the raw input data is fed directly into the network, and the network learns to extract features and make predictions, whereas decision trees and random forests typically require pre-processing of the data.

However, decision trees and random forests also have their own set of benefits, such as being interpretable, fast to train, and easy to implement. The choice of algorithm depends on the specific task and the characteristics of the data. Therefore MLP is a better algorithm for security purposes.

Chapter 4

Conclusion, Challenges and Future Work

4.1 Challenges and Future Directions

The optimization of MLPs for detecting PDF malware is a challenging task that requires careful consideration of the dataset, feature selection, hyperparameter tuning, and model evaluation. There are several challenges and future directions that should be considered in this area.

One challenge is the development of more advanced feature extraction techniques that can capture the complex and varied characteristics of PDF malware. For example, machine learning algorithms can be used to automatically extract features from PDF files, such as those based on deep learning techniques like convolutional neural networks (CNNs) or recurrent neural networks (RNNs) that can process the raw data directly, without the need for feature extraction.

Another challenge is the development of more sophisticated models that can handle the complexity of PDF files and detect more advanced forms of malware. For example, generative adversarial networks (GANs) can be used to generate realistic PDF files that contain hidden malware, which can be used to train and test detection models.

Furthermore, the integration of human expert knowledge and domain expertise can also enhance the performance of PDF malware detection systems. Human experts can provide insights into the characteristics of PDF malware that can be difficult to capture using machine learning techniques alone. For example, human experts can identify patterns in the content or structure of PDF files that are indicative of malicious activity.

Finally, the development of robust and interpretable models is also an important direction for future research. Robust models can handle adversarial attacks that are designed to evade detection, while interpretable models can provide insights into the features and patterns that are driving the model's predictions.

4.2 Conclusion

In conclusion, the optimization of MLPs for detecting PDF malware is a challenging and important area of research in machine learning. PDF malware is a growing threat that can have

serious consequences for individuals, organizations, and society as a whole. MLPs can be effective in detecting PDF malware by leveraging the power of machine learning to identify patterns in the content, structure, and metadata of PDF files. The optimization of MLPs involves several steps, including dataset curation, feature selection, hyperparameter tuning, and model evaluation. There are several challenges and future directions in this area, including the development of more advanced feature extraction techniques, the use of more sophisticated models, the integration of human expertise, and the development of robust and interpretable models. Overall, the optimization of MLPs for detecting PDF malware has the potential to make a significant impact in the fight against cybercrime and enhance the security of digital systems.

References

[1]ISO Central Secretary, “Document management — portable document format — part 1: Pdf 1.7,” International Organization for Standardization, Geneva, CH, Standard ISO/IEC TR 32000-1:2008, 2008. [Online]. Available: https://www.iso.org/standard/51502.html

[2] Ö. A. Aslan and R. Samet, “A comprehensive review on malware detection approaches,” IEEE Access, vol. 8, pp. 6249–6271, 2020. doi: 10.1109/ACCESS.2019.2963724

[3] S. Cesare, Y. Xiang, and W. Zhou, “Malwise—an effective and efficient classification system for packed and polymorphic malware,” IEEE Transactions on Computers, vol. 62, no. 6, pp. 1193–1206, 2012. doi: 10.1109/TC.2012.65

[4] D. Maiorca, B. Biggio, and G. Giacinto, “Towards adversarial malware detection: Lessons learned from PDF-based attacks,” ACM Computing Surveys (CSUR), vol. 52, no. 4, pp. 1–36, 2019. doi: 10.1145/3332184

[5] Q. Zhang and D. S. Reeves, “MetaAware: Identifying metamorphic malware,” in Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), Dec. 2007. doi: 10.1109/ACSAC.2007.9. ISSN 1063-9527 pp. 411–420.

[6]. Maryam Issakhani, Princy Victor, Ali Tekeoglu, and Arash Habibi Lashkari1, “PDF Malware Detection Based on Stacking Learning”, The International Conference on Information Systems Security and Privacy, February 2022

[7]. Chandran, P. P., Hema, R. N., & Jeyakarthic, M. (2022). Invasive weed optimization with stacked long short-term memory for PDF malware detection and classification. International Journal of Health Sciences, 6(S5), 4187–4204. https://doi.org/10.53730/ijhs.v6nS5.9540

[8]. Ekholm, Oscar. "Increased evasion resilience in modern PDF malware detectors: Using a more evasive training dataset." (2022).

[9]. Yerima, Suleiman Y., Abul Bashar, and Ghazanfar Latif. "Malicious PDF detection Based on Machine Learning with Enhanced Feature Set." 2022 14th International Conference on Computational Intelligence and Communication Networks (CICN). IEEE, 2022.

[10]. Abu Al-Haija, Q., A. Odeh, and H. Qattous. "PDF Malware Detection Based on Optimizable Decision Trees. Electronics 2022, 11, 3142." (2022).

[11]. Jiang, Tian, et al. "Application of deep reinforcement learning in attacking and protecting structural features-based malicious PDF detector." Future Generation Computer Systems 141 (2023): 325-338.

[12]. D’hooge, Laurens, et al. "Castles Built on Sand: Observations from Classifying Academic Cybersecurity Datasets with Minimalist Methods."

[13] Shalaginov, A., Banin, S., Dehghantanha, A., Franke, K. (2018). Machine Learning Aided Static Malware Analysis: A Survey and Tutorial. In: Dehghantanha, A., Conti, M.,

Dargahi, T. (eds) Cyber Threat Intelligence. Advances in Information Security, vol 70. Springer, Cham. https://doi.org/10.1007/978-3-319-73951-9_2

[14] VXunderground (VXheaven) https://www.vx-underground.org

[15] VirusShare https://virusshare.com

[16] Mahmoud Abdelsalam, Maanak Gupta, and Sudip Mittal. 2021. Artificial Intelligence Assisted Malware Analysis. In Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-physical Systems (SAT-CPS'21), April 28, 2021, Virtual Event, USA. ACM, New York, NY, USA, 3 pages. https: //doi.org/10.1145/3445969.3450433

[17] Michael R. Smith, Nicholas T. Johnson, Joe B. Ingram, Armida J. Carbajal, Bridget I. Haus, Eva Domschot, Ramyaa Ramyaa, Christopher C. Lamb, Stephen J. Verzi, and W. Philip Kegelmeyer. 2020. Mind the Gap: On Bridging the Semantic Gap between Machine Learning and Malware Analysis. In Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security (AISec'20). Association for Computing Machinery, New York, NY, USA, 49–60. https://doi.org/10.1145/3411508.3421373

[18] Peng Li, Limin Liu, Debin Gao, and Michael K. Reiter. 2010. On Challenges in Evaluating Malware Clustering. In Recent Advances in Intrusion Detection, Somesh Jha, Robin Sommer, and Christian Kreibich (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 238–255

[19] Roberto Perdisci and ManChon U. 2012. VAMO: Towards a Fully Automated Malware Clustering Validity Analysis. In Proceedings of the 28th Annual Computer Security Applications Conference (Orlando, Florida, USA) (ACSAC 2012). Association for Computing Machinery, New York, NY, USA, 329–338.]

[20] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D. Joseph, and J. D. Tygar. 2015. Better Malware Ground Truth: Techniques for Weighting Anti-Virus Vendor Labels. In Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security (Denver, Colorado, USA). Association for Computing Machinery, New York, NY, USA, 45–56.

[21] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. AVclass: A Tool for Massive Malware Labeling. In Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses (Lecture Notes in Computer Science), Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquín García-Alfaro (Eds.), Vol. 9854. Springer, 230–253.

[22] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. 2011. Malware Images: Visualization and Automatic Classification. In Proceedings of the 8th International Symposium on Visualization for Cyber Security (Pittsburgh, Pennsylvania, USA) (VizSec '11). Association for Computing Machinery, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/2016904.2016908

[23] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. 2018. Microsoft Malware Classification Challenge. CoRR abs/1802.10135 (2018).

[24] Hyrum S. Anderson and Phil Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. CoRR abs/1804.04637 (2018).

[25] Giorgio Severi, Tim Leek, and Brendan Dolan-Gavitt. 2018. Malrec: Compact fulltrace malware recording for retrospective deep analysis. In Detection of Intrusions and Malware, and Vulnerability Assessment - 15th International Conference, DIMVA 2018, Proceedings (Lecture Notes in Computer Science). Springer-Verlag, 3–23

[26] Marco Ramilli. 2016 (Accessed February 2020). Malware Training Sets: a machine learning dataset for everyone. https://marcoramilli.com/2016/12/16/malwaretraining-sets-a-machine-learning-dataset-for-everyone/

[27] Ferhat Özgür Çatak and Ahmet Faruk Yazi. 2019. A Benchmark API Call Dataset for Windows PE Malware Classification. CoRR abs/1905.01999 (2019)

[28] TheZoo - A live malware repository. https://thezoo.morirt.com/

[29] VirusTotal. https://www.virustotal.com

[30] Malware detection – make your own malware security system, in association with meraz'18 malware security partner Max Secure Software. https://www.kaggle.com/c/malware-detection

[31] R. Vyas, X. Luo, N. McFarland, and C. Justice. 2017. Investigation of malicious portable executable file detection on the network using supervised learning techniques. In 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). 941–946.

[32] "What are TPUs and how do they work?" by Sara Giorgetti, published on the OpenAI blog.

[33] "Cloud TPU" on the Google Cloud website.

[34] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, Charles Nicholas "Malware Detection by Eating a Whole EXE" https://doi.org/10.48550/arXiv.1710.09435

[35] Y. Oyama, T. Miyashita, and H. Kokubo. 2019. Identifying Useful Features for Malware Detection in the Ember Dataset. In 2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW). 360–366

[36] D. Stevens, "Malicious PDF documents explained," IEEE Security & Privacy, vol. 9, no. 1, pp. 80–82, Jan 2011. doi: 10.1109/MSP.2011.14

[37] T. Bienz and R. Cohn, Portable Document Format Reference Manual. Addison-Wesley Publishing Company, 1993. ISBN 0201626284, 9780201626285

[38] C. Carmony, X. Hu, H. Yin, A. V. Bhaskar, and M. Zhang, "Extract me if you can: Abusing PDF parsers in malware detectors," in Network and Distributed System Security (NDSS) Symposium, ser. NDSS, 2016. doi: 10.14722/NDSS.2016.23483. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2017/09/extract-me-if-you-can-abusing-pdf-parsers-malware-detectors.pdf

[39] ——, “Practical evasion of a learning-based classifier: A case study,” in 2014 IEEE Symposium on Security and Privacy, May 2014. doi: 10.1109/SP.2014.20. ISSN 2375-1207 pp. 197–211.

[40] Microsoft Defender Security Research Team, “Taking apart a double zero-day sample discovered in joint hunt with eset,” https://www.micr osoft.com/security/blog/2018/07/02/taking-apart-a-double-zero-day-sa mple-discovered-in-joint-hunt-with-eset/, 2018, accessed: 2022-03-07.

[41] Adobe, “Security updates available for Adobe Acrobat and Reader | apsb18-09,” https://helpx.adobe.com/security/products/acrobat/apsb18 -09.html#VulnerabilityDetails, 2019, accessed: 2022-03-07.

[42] Cybersecurity, Kaspersky Enterprise, “Machine learning for malware detection,” https://media.kaspersky.com/en/enterprise-security/Kas persky-Lab-Whitepaper-Machine-Learning.pdf, 2018.

[43] Maryam Issakhani, Princy Victor, Ali Tekeoglu, and Arash Habibi Lashkari1, “PDF Malware Detection Based on Stacking Learning”, The International Conference on Information Systems Security and Privacy, February 2022

Appendix A

Training of MLP

```
[ ]: # import basic libraries
     import numpy as np
     import pandas as pd
```

```
[ ]: # reading an excel file in pandas
     dataframe = pd.read_csv("sample_dataset/PDFMalware2022.csv")
```

```
[ ]: dataframe.head()
```

```
[ ]:                                        Fine name pdfsize metadata size \
     0                                                    8.0        180.0
     aedaf3c5428a2e3ba600c44b96ad78dfdf8ed76e7df129…
     1                                                   15.0        224.0
     fe767fb2584a10c010626263ea950643ac25f6ca24628f…
     2                                                    4.0        468.0
     544c5223ee301affad514b6fa585b3191625aba0a7222b…
     3                                                   17.0        250.0
     669772e626deccb9cfb7eb6a61e13d248d0ea08f1abe15…
     4                                                    7.0        252.0
     e434c884f45a691b0bf33d765f61794007eb0b8bb9f590…
   pages xref Length title characters isEncrypted embedded files images \
      0 1.0   11.0  0.0   0.0   0.0   0 1   0.0   20.0  7.0   0.0   0.0   0
     2   2.0      13.0  16.0  0.0   0.0   0
     3   1.0      15.0  0.0   0.0   0.0   0 4   3.0   16.0  45.0  0.0   0.0
            0

       text … AA OpenAction Acroform JBIG2Decode RichMedia launch \
     0   No … 0    1     0     0     0     0
     1   No … 0    0     1     0     0     0
     2   Yes … 0   1     0     0     0     0
     3   No … 0    1     1     0     0     0
     4   Yes … 0   1     0     0     0     0

       EmbeddedFile XFA Colors    Class
     0            0 0   0.0 Malicious
     1            8 1   0.0 Malicious
     2            0 0   0.0 Malicious
     3            0 0   0.0 Malicious
     4            0 0   0.0 Malicious

     [5 rows x 33 columns]
```

```
[ ]: dataframe.describe()
```

[ ]:
```
          pdfsize metadata size       pages   xref Length \
count 10025.000000 10025.000000 10025.000000 10025.000000
mean     87.209476   334.099352    3.398105  2739.220549
std     444.197122  1565.853177   11.902471 18139.229396
min      -1.000000    -1.000000   -1.000000    -1.000000
25%       9.000000   180.000000    1.000000    12.000000
50%      36.000000   265.000000    1.000000    21.000000
75%      80.000000   319.000000    2.000000    77.000000
max   23816.000000  77185.000000  595.000000  263987.000000
       title characters isEncrypted embedded files stream \

count   10025.000000 10025.000000 10025.000000 10023.000000
mean       51.477207   -0.020848    -0.006484   17.341215
std      1354.640037    0.206789     0.257098   35.330169
min        -1.000000   -1.000000    -1.000000   -1.000000
25%         0.000000    0.000000     0.000000    2.000000
50%         0.000000    0.000000     0.000000    4.000000
75%        13.000000    0.000000     0.000000   18.000000
max     76993.000000    4.000000     5.000000  812.000000

           trailer     encrypt      ObjStm      Colors
count 10023.000000 10023.000000 10023.000000 10023.000000
mean      1.203532   -0.043500    1.516811    2.087000
std       1.370455    0.256045    7.633485   58.178074
min      -1.000000   -1.000000   -1.000000   -1.000000
25%       1.000000    0.000000    0.000000    0.000000
50%       1.000000    0.000000    0.000000    0.000000
75%       2.000000    0.000000    0.000000    0.000000
max      46.000000    2.000000  600.000000 5682.000000
```

[ ]:
```
dataframe.head()
```

[ ]:
```
                                        Fine name pdfsize metadata size \
0                                                     8.0        180.0
aedaf3c5428a2e3ba600c44b96ad78dfdf8ed76e7df129…
1                                                    15.0        224.0
fe767fb2584a10c010626263ea950643ac25f6ca24628f…
2                                                     4.0        468.0
544c5223ee301affad514b6fa585b3191625aba0a7222b…
3                                                    17.0        250.0
669772e626deccb9cfb7eb6a61e13d248d0ea08f1abe15…
4                                                     7.0        252.0
e434c884f45a691b0bf33d765f61794007eb0b8bb9f590…
  pages xref Length title characters isEncrypted embedded files images \
     0 1.0   11.0  0.0   0.0   0.0   0 1   0.0   20.0  7.0   0.0   0.0   0
  2   2.0     13.0  16.0  0.0   0.0   0
  3   1.0     15.0  0.0   0.0   0.0   0
  4   3.0     16.0  45.0  0.0   0.0   0
```

```
  text … AA OpenAction Acroform JBIG2Decode RichMedia launch \
0   No … 0          1       0           0        0      0
1   No … 0          0       1           0        0      0
2 Yes … 0           1       0           0        0      0
3 No … 0            1       1           0        0      0
4 Yes … 0           1       0           0        0      0

 EmbeddedFile XFA Colors    Class
0            0 0  0.0 Malicious
1            8 1  0.0 Malicious
2            0 0  0.0 Malicious
3            0 0  0.0 Malicious
4            0 0  0.0 Malicious
[5 rows x 33 columns]
```

[ ]:
```
# Preprocessing the Dataset
#removing useless columns based on the statistical information
dataframe.shape #dimmensions
```

[ ]:
```
(10026, 33)
```

[ ]:
```
# replacing the file name with the index of the dataframe
dataframe['Fine name'] = dataframe.index # easiest possible approach to make them unique
```

[ ]:
```
dataframe.head()
```

[ ]:
```
Fine name pdfsize metadata size pages xref Length title characters \
0 0     8.0   180.0 1.0   11.0  0.0 1 1     15.0  224.0 0.0   20.0  7.0
2         2   4.0   468.0 2.0   13.0  16.0
3         3   17.0  250.0 1.0   15.0  0.0
4         4   7.0   252.0 3.0   16.0  45.0

   isEncrypted embedded files images text … AA OpenAction Acroform \
0 0.0   0.0   0 No … 0   1     0 1   0.0   0.0   0 No … 0   0
  1 2   0.0   0.0   0 Yes … 0  1     0 3   0.0   0.0   0 No … 0
  1     1
 4         0.0           0.0      0 Yes … 0          1       0

JBIG2Decode RichMedia launch EmbeddedFile XFA Colors  Class
0           0 0     0     0 0   0.0 Malicious
1           0 0     0     8 1   0.0 Malicious
2           0 0     0     0 0   0.0 Malicious
3           0 0     0     0 0   0.0 Malicious

4           0 0     0     0 0   0.0 Malicious

[5 rows x 33 columns]
```

[ ]:
```
means  = dataframe.mean()
to_drop = means[means<0].index
```

```
<ipython-input-36-26b5073bf24c>:1: FutureWarning: The default value
of numeric_only in DataFrame.mean is deprecated. In a future version,
it will default to False. In addition, specifying 'numeric_only=None'
is deprecated. Select only valid columns or specify the value of
numeric_only to silence this warning. means = dataframe.mean()
```

[ ]:
```
print(to_drop)
dataframe = dataframe.drop(to_drop,axis=1)
print(dataframe.columns)
```

```
Index(['isEncrypted', 'embedded files', 'encrypt'], dtype='object')
Index(['Fine name', 'pdfsize', 'metadata size', 'pages', 'xref
Length', 'title characters', 'images', 'text', 'header', 'obj',
                                                        'endobj',
  'stream', 'endstream', 'xref', 'trailer', 'startxref', 'pageno',
      'ObjStm', 'JS', 'Javascript', 'AA', 'OpenAction', 'Acroform',
      'JBIG2Decode', 'RichMedia', 'launch', 'EmbeddedFile', 'XFA',
      'Colors', 'Class'],
     dtype='object')
```

[ ]:
```
# # checking the dtype of columns
# # Select columns with dtype 'object'
# object_columns = dataframe.select_dtypes(include=['object'])
# # print(object_columns)
# # Drop the columns
# dataframe = dataframe.drop(object_columns, axis=1)
# dataframe.head()
```

[ ]:
```
# implementing machine learing algorithm
from sklearn.model_selection import train_test_split
dataframe = dataframe.dropna()
dataframe['Class'] = dataframe['Class'].replace({'Malicious': 1, 'Benign': 0})
X = dataframe.drop('Class', axis=1)

y = dataframe['Class']

object_columns = X.select_dtypes(include=['object'])
# print(object_columns)
# Drop the columns
X = X.drop(object_columns, axis=1)
```

```
print(X.head())



# print(y)
# y = y.reset_index(drop=True)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2,␣
↪random_state= 42)
```

```
Fine name pdfsize metadata size pages xref Length title characters \
 0         0   8.0   180.0 1.0   11.0  0.0
 1         1   15.0  224.0 0.0   20.0  7.0
 2         2   4.0   468.0 2.0   13.0  16.0
 3         3   17.0  250.0 1.0   15.0  0.0
 4         4   7.0   252.0 3.0   16.0  45.0

   stream trailer ObjStm Colors
0 3.0   1.0   0.0   0.0 1 9.0
  1.0   0.0   0.0 2 3.0   1.0
  0.0   0.0 3 2.0   1.0   0.0
  0.0
4    4.0     1.0    0.0    0.0
```

[ ]:
```
print(y.unique())
```

```
[1 0]
```

[ ]:
```
# implementing machine learning model in python
from sklearn.neural_network import MLPClassifier

# Define the model
model = MLPClassifier(hidden_layer_sizes=(6,100,50,10),max_iter=5000000 ,␣
↪alpha=1e-10,
                      solver='adam', verbose=10,␣
↪random_state=1,learning_rate_init=.0001)

# Train the model
model.fit(X_train, y_train)

# Make predictions
predictions = model.predict(X_test)

print((np.sum(predictions==y_test)/len(predictions))*100)
```

```
Iteration 1, loss = 9.11871321
Iteration 2, loss = 0.95289272
```

```
Iteration 3, loss = 0.84945731
Iteration 4, loss = 0.83282700
Iteration 5, loss = 0.81996741
Iteration 6, loss = 0.80703455
Iteration 7, loss = 0.79462416
Iteration 8, loss = 0.78335563
Iteration 9, loss = 0.77675159
Iteration 10, loss = 0.77221580
Iteration 11, loss = 0.76454551
Iteration 12, loss = 0.75662850
Iteration 13, loss = 0.73817340
Iteration 14, loss = 0.73855514
Iteration 15, loss = 0.72396001
Iteration 16, loss = 0.72525145
Iteration 17, loss = 0.72785319
Iteration 18, loss = 0.71423694
Iteration 19, loss = 0.71072084
Iteration 20, loss = 0.69607157
Iteration 21, loss = 0.69161742
Iteration 22, loss = 0.68370341
Iteration 23, loss = 0.67940726
Iteration 24, loss = 0.67794917
Iteration 25, loss = 0.66942196
Iteration 26, loss = 0.66593699
Iteration 27, loss = 0.66379387
Iteration 28, loss = 0.65203520
Iteration 29, loss = 0.65413469
Iteration 30, loss = 0.64603790
Iteration 31, loss = 0.64242178
Iteration 32, loss = 0.64649772
Iteration 33, loss = 0.62901734
Iteration 34, loss = 0.61703872
Iteration 35, loss = 0.62438432
Iteration 36, loss = 0.61343330
Iteration 37, loss = 0.59883955
Iteration 38, loss = 0.59910119
Iteration 39, loss = 0.59336769
Iteration 40, loss = 0.58197049
Iteration 41, loss = 0.58657537
Iteration 42, loss = 0.58356771
Iteration 43, loss = 0.56672245
Iteration 44, loss = 0.56719903
Iteration 45, loss = 0.56227133
Iteration 46, loss = 0.54922892
Iteration 47, loss = 0.54054469
Iteration 48, loss = 0.53382364
```

```
Iteration 49, loss = 0.53352346
Iteration 50, loss = 0.52666215
Iteration 51, loss = 0.52154725 Iteration 52, loss = 0.51042947
Iteration 53, loss = 0.50321544
Iteration 54, loss = 0.49832079
Iteration 55, loss = 0.48359526
Iteration 56, loss = 0.47009186
Iteration 57, loss = 0.45313203
Iteration 58, loss = 0.43306311
Iteration 59, loss = 0.43989371
Iteration 60, loss = 0.42011681
Iteration 61, loss = 0.41321519
Iteration 62, loss = 0.40695503
Iteration 63, loss = 0.40724181
Iteration 64, loss = 0.40333271
Iteration 65, loss = 0.39892181
Iteration 66, loss = 0.39530011
Iteration 67, loss = 0.39465271
Iteration 68, loss = 0.38575951
Iteration 69, loss = 0.38977782
Iteration 70, loss = 0.39241008
Iteration 71, loss = 0.38298531
Iteration 72, loss = 0.38488808
Iteration 73, loss = 0.37498073
Iteration 74, loss = 0.36855905
Iteration 75, loss = 0.37137417
Iteration 76, loss = 0.36366008
Iteration 77, loss = 0.35802508
Iteration 78, loss = 0.36120000
Iteration 79, loss = 0.35615484
Iteration 80, loss = 0.35119258
Iteration 81, loss = 0.35223679
Iteration 82, loss = 0.37421088
Iteration 83, loss = 0.35070015
Iteration 84, loss = 0.34418504
Iteration 85, loss = 0.34707818
Iteration 86, loss = 0.34654475
Iteration 87, loss = 0.34174237
Iteration 88, loss = 0.34065737
Iteration 89, loss = 0.33856552
Iteration 90, loss = 0.33817393
Iteration 91, loss = 0.33985001
Iteration 92, loss = 0.33868208
Iteration 93, loss = 0.34028122
Iteration 94, loss = 0.34792737
Iteration 95, loss = 0.33533680
```

```
Iteration 96, loss = 0.33220528
Iteration 97, loss = 0.32692242
Iteration 98, loss = 0.32474545
Iteration 99, loss = 0.32556744 Iteration 100, loss = 0.32585011
Iteration 101, loss = 0.33130359
Iteration 102, loss = 0.32627468
Iteration 103, loss = 0.34263454
Iteration 104, loss = 0.31722116
Iteration 105, loss = 0.31517071
Iteration 106, loss = 0.33748426
Iteration 107, loss = 0.34150354
Iteration 108, loss = 0.32112366
Iteration 109, loss = 0.33003322
Iteration 110, loss = 0.32560238
Iteration 111, loss = 0.32096569
Iteration 112, loss = 0.31789361
Iteration 113, loss = 0.32236922
Iteration 114, loss = 0.30937929
Iteration 115, loss = 0.31289201
Iteration 116, loss = 0.34638520
Iteration 117, loss = 0.30160586
Iteration 118, loss = 0.30057086
Iteration 119, loss = 0.31512145
Iteration 120, loss = 0.31651453
Iteration 121, loss = 0.30068118
Iteration 122, loss = 0.30150828
Iteration 123, loss = 0.29797487
Iteration 124, loss = 0.29781278
Iteration 125, loss = 0.29797281
Iteration 126, loss = 0.29519166
Iteration 127, loss = 0.29437750
Iteration 128, loss = 0.30358302
Iteration 129, loss = 0.29405439
Iteration 130, loss = 0.29621953
Iteration 131, loss = 0.28793387
Iteration 132, loss = 0.28869428
Iteration 133, loss = 0.28770160
Iteration 134, loss = 0.28398103
Iteration 135, loss = 0.28686016
Iteration 136, loss = 0.28962518
Iteration 137, loss = 0.30716241
Iteration 138, loss = 0.29932123
Iteration 139, loss = 0.28291992
Iteration 140, loss = 0.29396323
Iteration 141, loss = 0.28548987
Iteration 142, loss = 0.28986208
```

```
Iteration 143, loss = 0.30244561
Iteration 144, loss = 0.27565666
Iteration 145, loss = 0.28042302
Iteration 146, loss = 0.28392951
Iteration 147, loss = 0.29702190 Iteration 148, loss = 0.27929378
Iteration 149, loss = 0.27099194
Iteration 150, loss = 0.27095004
Iteration 151, loss = 0.27055895
Iteration 152, loss = 0.26893495
Iteration 153, loss = 0.26788807
Iteration 154, loss = 0.28987887
Iteration 155, loss = 0.26827556
Iteration 156, loss = 0.28343639
Iteration 157, loss = 0.26583491
Iteration 158, loss = 0.26359279
Iteration 159, loss = 0.27241453
Iteration 160, loss = 0.26286970
Iteration 161, loss = 0.26502490
Iteration 162, loss = 0.26238632
Iteration 163, loss = 0.26004744
Iteration 164, loss = 0.25699075
Iteration 165, loss = 0.25728841
Iteration 166, loss = 0.25852979
Iteration 167, loss = 0.25412399
Iteration 168, loss = 0.27238455
Iteration 169, loss = 0.26753600
Iteration 170, loss = 0.26919628
Iteration 171, loss = 0.25371573
Iteration 172, loss = 0.25141928
Iteration 173, loss = 0.25743960
Iteration 174, loss = 0.26240988
Iteration 175, loss = 0.27237598
Iteration 176, loss = 0.24975103
Iteration 177, loss = 0.24736685
Iteration 178, loss = 0.25089413
Iteration 179, loss = 0.25535965
Iteration 180, loss = 0.24412423
Iteration 181, loss = 0.24921267
Iteration 182, loss = 0.24651067
Iteration 183, loss = 0.24835918
Iteration 184, loss = 0.24753967
Iteration 185, loss = 0.25078694
Iteration 186, loss = 0.24331867
Iteration 187, loss = 0.23804045
Iteration 188, loss = 0.24079863
Iteration 189, loss = 0.23469860
```

```
Iteration 190, loss = 0.26091576
Iteration 191, loss = 0.23985501
Iteration 192, loss = 0.26267464
Iteration 193, loss = 0.23664145
Iteration 194, loss = 0.23127756
Iteration 195, loss = 0.24042043 Iteration 196, loss = 0.23021461
Iteration 197, loss = 0.23291935
Iteration 198, loss = 0.23060653
Iteration 199, loss = 0.22903553
Iteration 200, loss = 0.23347759
Iteration 201, loss = 0.22597181
Iteration 202, loss = 0.22527434
Iteration 203, loss = 0.22601266
Iteration 204, loss = 0.22625700
Iteration 205, loss = 0.22431090
Iteration 206, loss = 0.22271718
Iteration 207, loss = 0.22218772
Iteration 208, loss = 0.22606671
Iteration 209, loss = 0.21916377
Iteration 210, loss = 0.22933461
Iteration 211, loss = 0.23056713
Iteration 212, loss = 0.21882094
Iteration 213, loss = 0.21677561
Iteration 214, loss = 0.21962397
Iteration 215, loss = 0.21419946
Iteration 216, loss = 0.21913049
Iteration 217, loss = 0.21759535
Iteration 218, loss = 0.21444900
Iteration 219, loss = 0.23396053
Iteration 220, loss = 0.21926805
Iteration 221, loss = 0.21731662
Iteration 222, loss = 0.22647935
Iteration 223, loss = 0.21441907
Iteration 224, loss = 0.21120644
Iteration 225, loss = 0.21147581
Iteration 226, loss = 0.21145614
Iteration 227, loss = 0.20863352
Iteration 228, loss = 0.20386253
Iteration 229, loss = 0.21073107
Iteration 230, loss = 0.20439845
Iteration 231, loss = 0.22220631
Iteration 232, loss = 0.20870704
Iteration 233, loss = 0.20501992
Iteration 234, loss = 0.21169930
Iteration 235, loss = 0.20688394
Iteration 236, loss = 0.20802939
```

```
Iteration 237, loss = 0.21789063
Iteration 238, loss = 0.23596344
Iteration 239, loss = 0.21830213
Training loss did not improve more than tol=0.000100 for 10
consecutive epochs. Stopping.
```

```

```

```

```

```
96.05985037406484
```

[ ]:

[ ]:

Appendix B

Comparison of MLP with other Algorithms

```
[ ]: # pip install seaborn
```

```
[ ]: import pandas as pd import
numpy as np import
matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import
RepeatedStratifiedKFold from sklearn.linear_model
import LogisticRegression from sklearn.tree import
DecisionTreeClassifier from sklearn.ensemble import
RandomForestClassifier from sklearn.ensemble import
AdaBoostClassifier from xgboost import
XGBClassifier from sklearn.model_selection import
RandomizedSearchCV from sklearn.model_selection
import train_test_split as tts import lightgbm as
lgb import gc from sklearn.metrics import log_loss
from sklearn.metrics import roc_auc_score from
sklearn.metrics import accuracy_score from
sklearn.metrics import f1_score
# from sklearn.metrics import plot_confusion_matrix,
confusion_matrix from sklearn.neighbors import NearestNeighbors
import seaborn as sns
import calendar
from sklearn.decomposition import PCA from
sklearn.linear_model import LogisticRegression
from sklearn.model_selection import
train_test_split from sklearn.model_selection
import RandomizedSearchCV from
sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score,
mean_squared_error from sklearn.model_selection
import cross_validate from sklearn.metrics
import make_scorer from sklearn.metrics import
confusion_matrix import warnings
warnings.filterwarnings('ignore') sns.set()
import warnings
```

```
warnings.filterwarnings('ignore')
```

[ ]:
```
# reading an excel file in pandas
dataframe = pd.read_csv("sample_dataset/PDFMalware2022.csv")
```

[ ]:
```
dataframe.describe()
```

[ ]:
```
            pdfsize  metadata size         pages    xref Length  \
count 10025.000000 10025.000000 10025.000000 10025.000000
mean     87.209476   334.099352     3.398105  2739.220549
std     444.197122  1565.853177    11.902471 18139.229396
min      -1.000000    -1.000000    -1.000000    -1.000000
25%       9.000000   180.000000     1.000000    12.000000
50%      36.000000   265.000000     1.000000    21.000000
75%      80.000000   319.000000     2.000000    77.000000
max   23816.000000 77185.000000   595.000000 263987.000000

       title charactersisEncryptedembedded files       stream\
count    10025.000000 10025.000000  10025.000000 10023.000000
mean        51.477207    -0.020848     -0.006484    17.341215
std       1354.640037     0.206789      0.257098    35.330169
min         -1.000000    -1.000000     -1.000000    -1.000000
25%          0.000000     0.000000      0.000000     2.000000
50%          0.000000     0.000000      0.000000     4.000000
75%         13.000000     0.000000      0.000000    18.000000
max      76993.000000     4.000000      5.000000   812.000000

          trailer      encrypt       ObjStm       Colors
count 10023.000000 10023.000000 10023.000000 10023.000000
mean      1.203532    -0.043500     1.516811     2.087000
std       1.370455     0.256045     7.633485    58.178074
min      -1.000000    -1.000000    -1.000000    -1.000000
25%       1.000000     0.000000     0.000000     0.000000
```

```
50%       1.000000     0.000000     0.000000     0.000000
75%       2.000000     0.000000     0.000000     0.000000
max      46.000000     2.000000   600.000000  5682.000000
```

[ ]:
```
# replacing the file name with the index of the dataframe
dataframe['Fine name'] = dataframe.index # easiest possible approach to make␣
↪them unique
```

[ ]:
```
means  = dataframe.mean()
to_drop = means[means<0].index
```

[ ]:
```
print(to_drop)
dataframe = dataframe.drop(to_drop,axis=1)
print(dataframe.columns)
```

```
Index(['isEncrypted', 'embedded files', 'encrypt'],
dtype='object')
Index(['Fine name', 'pdfsize', 'metadata size', 'pages', 'xref
Length', 'title characters', 'images', 'text', 'header', 'obj',
                                                      'endobj',
       'stream', 'endstream', 'xref', 'trailer', 'startxref',
       'pageno',
       'ObjStm', 'JS', 'Javascript', 'AA', 'OpenAction',
       'Acroform',
       'JBIG2Decode', 'RichMedia', 'launch', 'EmbeddedFile', 'XFA',
       'Colors', 'Class'],
```

dtype='object')

[ ]:
```
#data preperation
dataframe = dataframe.dropna()
dataframe['Class'] = dataframe['Class'].replace({'Malicious': 1, 'Benign': 0})
X = dataframe.drop('Class', axis=1)

y = dataframe['Class']

object_columns = X.select_dtypes(include=['object'])
# print(object_columns)
# Drop the columns
X = X.drop(object_columns, axis=1)
print(X.head())



# print(y)
# y = y.reset_index(drop=True)


X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=.
↪2,random_state =42)
```

```
   Fine name  pdfsize  metadata size  pages  xref Length  title characters  \
0          0      8.0          180.0    1.0         11.0               0.0
1          1   15.0  224.0 0.0   20.0  7.0
2          2   4.0   468.0 2.0   13.0  16.0
3          3   17.0  250.0 1.0   15.0  0.0
4          4   7.0   252.0 3.0   16.0  45.0

  stream trailer ObjStm Colors
0 3.0   1.0   0.0   0.0 1 9.0
  1.0   0.0   0.0 2 3.0   1.0
  0.0   0.0 3 2.0   1.0   0.0
  0.0 4 4.0   1.0   0.0   0.0
```

[ ]:
```
from lazypredict.Supervised import
LazyClassifier from sklearn.model_selection
import train_test_split
```

[ ]:
```
clf = LazyClassifier(verbose=0,ignore_warnings=True,

custom_metric=None) models,predictions = clf.fit(X_train, X_test,

y_train, y_test) models.sort_values(by = ["Accuracy","F1
```

```
Score"],ascending = False)

100%|| 29/29 [00:10<00:00,  2.82it/s]
```

[ ]:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score
ExtraTreesClassifier	1.00	1.00	1.00	1.00
XGBClassifier	1.00	1.00	1.00	1.00
RandomForestClassifier	1.00	1.00	1.00	1.00
BaggingClassifier	1.00	1.00	1.00	1.00
AdaBoostClassifier	1.00	1.00	1.00	1.00
LGBMClassifier	1.00	1.00	1.00	1.00
DecisionTreeClassifier	1.00	1.00	1.00	1.00
KNeighborsClassifier	0.99	0.99	0.99	0.99
LabelPropagation	0.99	0.99	0.99	0.99
LabelSpreading	0.99	0.99	0.99	0.99
ExtraTreeClassifier	0.99	0.99	0.99	0.99
SVC	0.99	0.99	0.99	0.99
SGDClassifier	0.98	0.98	0.98	0.98
LogisticRegression	0.97	0.97	0.97	0.97
LinearDiscriminantAnalysis	0.97	0.97	0.97	0.97
RidgeClassifierCV	0.97	0.97	0.97	0.97
RidgeClassifier	0.97	0.97	0.97	0.97
CalibratedClassifierCV	0.97	0.97	0.97	0.97
LinearSVC	0.96	0.96	0.96	0.96
NuSVC	0.95	0.96	0.96	0.95
NearestCentroid	0.95	0.95	0.95	0.95
BernoulliNB	0.94	0.94	0.94	0.94
Perceptron	0.88	0.88	0.88	0.88
PassiveAggressiveClassifier	0.88	0.89	0.89	0.88
QuadraticDiscriminantAnalysis	0.82	0.83	0.83	0.81
GaussianNB	0.80	0.82	0.82	0.80
	0.56	0.50	0.50	0.40

```
DummyClassifier
```

Model	Time Taken
ExtraTreesClassifier	0.32
XGBClassifier	0.20
RandomForestClassifier	0.44
BaggingClassifier	0.08
AdaBoostClassifier	0.40
LGBMClassifier	0.21
DecisionTreeClassifier	0.02
KNeighborsClassifier	0.16
LabelPropagation	1.12
LabelSpreading	2.37
ExtraTreeClassifier	0.02
SVC	0.49
SGDClassifier	0.04
LogisticRegression	0.04
LinearDiscriminantAnalysis	0.04
RidgeClassifierCV	0.02
RidgeClassifier	0.02
CalibratedClassifierCV	1.00
LinearSVC	0.25
NuSVC	2.89
NearestCentroid	0.02
BernoulliNB	0.02
Perceptron	0.02
PassiveAggressiveClassifier	0.02
QuadraticDiscriminantAnalysis	0.02
GaussianNB	0.02
DummyClassifier	0.02

[ ]:

Appendix C

Comparison of Time taken by different algorithms