A Novel Evasive PDF Malware Detection Model Based on Stacking Learning

by

Maryam Issakhani

Bachelor of Software Engineering, Alzahra, 2019

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Computer Science MSC

Computer Science

Supervisor(s): Arash Habibi Lashkari, Ph.D., Computer Science

Examining Board: Rongxing Lu, Ph.D., Computer Science, Chair
Saqib Hakak, Ph.D., Computer Science,
Mohsen Mohammadi, Ph.D.,Engineering

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

December, 2021

© Maryam Issakhani, 2021

Abstract

Over the last few years, Portable Document Format (PDF) has become the most popular content presenting format among users due to its extraordinarily flexible and easy-to-work features. However, advanced PDF features such as Javascript injection or file embedding make them an attractive target to exploit by attackers. Due to the complex PDF structure and sophistication of attacks, traditional detection approaches such as Anti-Viruses are ineffective as they rely on signature-based techniques. Various research works take a different direction and attempt to utilize AI technologies such as Machine Learning (ML) and Deep Learning (DL) to detect malicious PDF files. Despite the results from the research communities, the evasive malicious PDF files are still a security threat. This research attempts to address this gap by proposing a novel framework that stacks ML models for detecting an evasive malicious PDF file. In addition to that, we evaluated our solution by using two different datasets, Contagio and a newly generated evasive PDF dataset. In the first evaluation, we achieved accuracy and F1-score of 99.89% and 99.86%, which is better than the performance of existing models. Moreover, we re-evaluated our framework using our new evasive PDF dataset as an improved version of Contagio to verify our solution further. As a result, we achieved 98.69% and 98.77% as accuracy and F1-scores, demonstrating the effectiveness of our approach to improve the results. Experimental results along with the new dataset proved that our model could be applied in practice.

Table of Contents

Abstract ii

Table of Contents iii

List of Tables vii

List of Figures viii

1 Introduction 1

1.1 Introducing PDF Files And Security Threats 1

1.2 Summary Of Contributions 4

1.3 Thesis Organization 4

1.4 Concluding Remarks 5

2 Literature Review 7

2.1 Document-Based Malware 7

2.2 PDF Malware 12

2.2.1 Exploiting PDF Features For Delivering Malware 14

2.2.2 PDF Obfuscation 16

2.2.3 PDF Malware Detection 17

2.2.3.1 Static Analysis 18

2.2.3.2 Dynamic Analysis 22

2.3 Phishing Attack Using PDF 25

2.3.1 Introduction To Phishing Attack 25

2.3.2 PDF-Based Phishing 26
2.3.3 Defending Against Phishing Attacks 27
2.4 Comparing PDF Malware Detection Techniques 29
2.4.1 Machine Learning Solutions Vs. Deep Learning Solutions . . . 30
2.5 Remaining Concerns 31
2.6 Concluding Remarks 35

3 Proposed Approach **36**
3.1 General Overview 36
3.2 Feature Extraction 37
3.3 Detection Model 41
3.3.1 Overview Of Stacking 41
3.3.2 Training And Testing Stacking-Based Classifiers 42
3.3.3 Proposed Stacking Architecture 42
3.3.3.1 Number Of Stacking Levels 43
3.3.3.2 Number Of Base Learners 43
3.3.3.3 Types Of Base Learners 43
3.3.3.4 Number Of Meta Learners 45
3.3.3.5 Types Of Meta Learners 45
3.3.3.6 Dataset Split 47
3.4 Handling The Overfitting Issue 47
3.5 Concluding Remarks 49

4 Implementation **50**
4.1 Overview 50
4.2 Experimental Setup 50
4.3 Feature Extraction 51
4.3.1 Tools And Libraries 52

4.3.2 Coding Details 52
4.4 Pre-Processing 53
4.4.1 Manual Pre-Processing 54
4.4.2 Python Pre-Processing 54
4.4.3 Tools And Libraries 54
4.4.4 Coding Detail 55
4.5 Detection Model Using Python 55
4.5.1 Tools And Libraries 55
4.5.2 Coding Detail 55
4.6 Concluding Remarks 56

5 Generating New Dataset **57**
5.1 Evaluating The Available Dataset 58
5.2 Creating An Evasive PDF Dataset 60
5.3 Evaluating The New Dataset 62
5.4 Concluding Remarks 64

6 Experiments and Results **66**
6.1 Data Repository 66
6.1.1 Contagio Dataset Distribution 66
6.1.2 Evasive PDF Dataset Distribution 67
6.2 Finalizing The Proposed Model 67
6.2.1 Dataset Split 68
6.2.2 Tuning Base-Learner Parameters 68
6.2.3 Finalizing Learning Detection Model 69
6.3 Evaluating The Proposed Model 71
6.3.1 Comparing Proposed Model With Relevant Studies 72
6.3.2 Evaluating The Proposed Model Using New Dataset 72

6.4 Other Findings 73
6.5 Concluding Remarks 74

7 Conclusion And Future Work **75**
7.1 Conclusion 75
7.2 Future Work 76

Bibliography **83**

A This is Appendix A **84**

Vita

List of Tables

2.1 Relevant Research Work Comparison 32

5.1 Available dataset resources for generating our new dataset 60

5.2 Our dataset specifications 62

5.3 Variance and Standard Deviation Comparison Results 64

6.1 Contagio Dataset distribution 67

6.2 Our generated csv data file from Contagio specifications 67

6.3 dataset distribution 67

6.4 Base-Learner Results 69

6.5 Proposed model results with different meta-learners 70

6.6 Comparison of our work with relevant studies 72

6.7 base-learner results on the new dataset 73

6.8 Proposed model results on our dataset 73

List of Figures

2.1 PDF Internal Structure 13

2.2 PDF Internal Structure/Encrypted Object 14

2.3 PDF Phishing With a Hidden Malicious URL Taken From [10] 28

2.4 PDF Phishing With Static Image Resembling a Video Taken From [10] 28

2.5 Taxonomy Of PDF Malware 35

3.1 Layout of the proposed architecture 37

3.2 Proposed Detection Model 48

4.1 Implementation Overview 51

4.2 Code Snippet For Extracting Features Using PyMuPDF 53

4.3 Code Snippet For Extracting PDFid Features 53

4.4 Pre-Processing The Feature values 55

4.5 Code Snippet For Stacking Implementation 56

6.1 Base learner Results Visualization 70

6.2 Final proposed stacking architecture 71

6.3 Metamodel Results Visualization 71

Chapter 1

Introduction

1.1 Introducing PDF Files And Security Threats

Over the years, PDF has been the most widely used document format due to its portability and reliability. Unlike the rest of the document formats, such as doc files that may present the content differently on different platforms, PDF files are platform-independent. Despite the appearance, it has a very complex structure with a combination of binary and ASCII data, and it can be considered a whole programming language of its own, which gets interpreted and executed upon being viewed. This makes a massive difference with simple text files that only display the content upon being opened. Moreover, PDFs support various features and elements such as embedding multimedia files, JavaScript functionalities and adding system commands inside the PDF. These features contributed a lot to the flexibility and efficiency of PDFs.

PDF popularity and its supported features, specifically JavaScript, have allowed attackers to exploit them in numerous ways. Spreading malware through PDFs has become a common technique in today's online world, where documents are increasingly sent in PDF format compared to paper sheets. In 2008, [16] analyzed the

critical PDF features that an attacker can misuse to deliver a malicious payload. The paper demonstrates that attackers have various options to utilize and combine these features together to launch an attack. To further support the high attack potential through PDF, they highlighted two successful proof of concepts that evaded anti-virus detection. In addition to that, they provided some available security measures to minimize this risk. The majority of PDF attacks target Adobe Reader, one of the most popular PDF readers among users. Despite being a powerful PDF viewer, it has many vulnerabilities, often the target for various PDF exploitation. [37] listed 27 main types of vulnerabilities found in Adobe Reader, such as API overflow and memory corruption. These vulnerabilities are often exploited by attackers, mainly through using Javascript.

A plethora of researches, namely [50][34], discussed that traditional automated detection methods are usually inefficient for malicious PDF detection as they are mostly signature-based, which allows the malware authors to evade them sometimes, even just with basic obfuscation. Therefore, PDF infection detection still requires manual analysis using different tools, making them the perfect tool for attackers, as they easily bypass common anti-virus software such as McAfee, Avast, and Norton. Moreover, most users only associate danger with executable files and do not think of files such as .doc and .pdf as being potentially malicious, according to a study in 2014 titled "PDF Malware: Can PDFs Contain Viruses?" from "www.cyren.com".

As mentioned before, PDF can be much more than just content delivery, so they are an increasingly attractive tool for attackers to exploit. We can divide PDF malware into two broad categories:

i. Phishing: Phishing mainly targets users with the intent to trick them into disabling their security features or clicking on a malicious link inside the PDF. It can be integrated with other techniques and features to deliver an attack in many ways. For example, an attacker may embed a malicious payload inside the PDF and

then modify the warning dialog box displayed on Adobe Reader to fool the user into ignoring the warning and let the malicious payload execute. PDF phishing attacks have been widely used recently as an alternative to email spamming. It is one of the most effective approaches used by attackers due to the general public trust in PDFs and the inability of anti-viruses to detect harmful links hidden inside of them.

ii. Exploits: An exploit is a program or a piece of code that finds and takes advantage of a security flaw in the system or a PDF viewing application and does not need to trick the user into initiating it. PDF exploits can be divided into JavaScript-based exploits and non-JavaScript-based ones.

- JavaScript-based exploits: JavaScript is supported in PDF as it utilizes this feature when a link or button is clicked or while filling out a form inside the PDF. JavaScript is often the workhorse that attackers use to set up and execute attacks due to its flexibility and ease of use. There are various ways malware embedded PDFs can use JavaScript, such as connecting to a malicious URL, executing embedded shellcode, and launching malicious embedded files.

- Non-JavaScript-based exploits: A few cases observed are alternatives to JavaScript-based exploits, such as embedding media files and flash objects in PDF files.

There are various easy preventive measures suggested by [1], which can significantly help reduce the risk of malware-embedded PDFs, such as disabling features like JavaScript and macros and keeping document-viewing software updated. However, as malware continues to evolve, the benefits these features provide can be applied only temporarily. In addition, disabling these features can also negatively impact the end-users experience while using the product for legitimate reasons. Considering the above facts, this research proposes an evasive PDF malware detection using stacking machine learning and deep learning solutions. This proposed solution scans raw PDF files and extracts a set of features, and then classifies them as either benign or

malicious.

1.2 Summary Of Contributions

In this work, the main contributions are as follows:

1. We proposed a novel approach based on Machine Learning Stacking to predict if a PDF is malicious, resulting in improved accuracy and F1-score compared with the relevant works. (*Con1*)

2. We proposed and presented a general taxonomy of possible attack surface deliveries through PDF files. (*Con2*)

3. We analyzed the shortcomings of the "Contagio" PDF dataset, one of the most popular datasets used for PDF malware detection. (*Con3*)

4. We addressed most of the specified flaws in the Contagio dataset by combining it with the VirusTotal dataset. As a result, we generated a more comprehensive and stable dataset with more reliable results closer to the real-world distribution of the data. (*Con4*)

5. We developed the proposed features extraction solution that takes a PDF folder path as an input and extracts 32 features from each input file. (*Con5*)

1.3 Thesis Organization

Below is the overview of the thesis organization :

1. In chapter 1, the PDF document format is briefly introduced, and its main relevant security threats are analyzed. We next discussed common countermeasures applied by users to reduce the PDF malware potential risks.

2. In chapter 2, we provided an overview of the PDF internal structure to give more insight about PDF as a page description programming language. Furthermore, we categorized the most commonly employed techniques to detect malicious PDF files and examined a set of different research works and their proposed approaches regarding each category.

3. In chapter 3, we highlighted the seriousness of PDF malware attacks and the significance of an effective detection approach. In addition to that, we discussed the main motivations to do this work by reviewing the challenges of the current detection methods.

4. In chapter 4, we proposed a comprehensive set of 37 features extracted from each PDF file and designed a novel detection approach to classify the PDF files based on the extracted features. Moreover, a detailed justification is provided for the selected design and its potential effectiveness.

5. In chapter 5, we tested our proposed model on two different datasets and examined the results using relevant measurement metrics. Further, we provided a table to compare the results with relevant research works.

6. In chapter 6, we analyzed the results obtained in chapter 5 to demonstrate the improvement of our proposed approach. Next, we highlighted some of the baseline dataset issues used by most of the research works.

7. In chapter 7, we discussed some of the reminding issues in this domain and the future directions of this study.

1.4 Concluding Remarks

As PDF malware threats are pretty tricky and complex to detect, developing an effective solution is challenging. In this chapter, we listed our main contributions

and presented thesis organization. The next chapter will elaborate and analyze the PDF features, attack surfaces, and the primary detection approaches explored in different research works.

Chapter 2

Literature Review

This chapter provides an overview of document-based malware detection and discusses phishing techniques regarding PDF files. We also present a general background about the PDF document structure, and take a closer look at different PDF features and how attackers can maliciously exploit them. Lastly, we analyze and compare main detection approaches and the relevant works in terms of reliability and challenges and briefly introduce standard tools for PDF inspection.

2.1 Document-Based Malware

Documents have become an increasingly popular attack vector for malware authors as more sophisticated features are added to documents to improve the user experience. According to Barracuda's report, [14], a recent email analysis revealed that 48 percent of all malicious files detected in the last 12 months were in the form of a document.

One of the most common attack surfaces exploited in official documents is the macro feature. Macros are advanced features of document files written in Visual Basic (VBA) programming language created to make some tasks more accessible. According to Microsoft, creating Macros is used to "automate frequently used tasks".However,

they can gain access to the operating and file system of the computer, giving the attacker the ability to delete files, upload data, and even download spyware.

Auto open macros are macros that get executed upon opening the document every time, and attackers take advantage of this feature to execute their malicious code on the user's system as soon as the document is opened, leaving the end-user little to no time to react. This creates a situation where end users must rely on security features either included in the document software to prevent automatic macros from running or an anti-virus capable of detecting macros maliciously accessing the machine.

One of the common ways that attackers exploit macros is by embedding a Base64-encoded shellcode using certain API functions [26]. Moreover, it can connect to a malicious URL that initiates the download of a malware payload to the victim's machine [39].

Detecting malicious macro codes has been researched in various studies; in [15], opcodes are extracted from the p-code on multiple pieces of macro code, and n-gram sequences of the extracted opcodes are used as features. The features are then selected based on their rank and applied KNN as the base classifier. The model was then evaluated using different neighbors(n), top features(l), and N-gram variations, and results proved that 1-gram feature selection achieved better overall accuracy than 2-grams.

In another work [40], a linguistic-based macros detection method is proposed where new malware families are identified using the word frequency in the source code. First, the macro samples of xls, ppt, doc, xlsx, pptx, docx extensions were collected from VirusTotal, and the macros inside these files were extracted. The extracted macros are then used for the corpora creation by extracting words and replacing special characters such as double quotes, commas, asterisks, etc., into blank space. Then, based on the predefined patterns, the obfuscated words are converted into one word for a better detection rate, and a dictionary is created based on the unique

words from which the feature vectors are extracted. Finally, the features are passed to the SVM model for training, and results showed that the proposed model with malicious corpora dictionary increased the detection rate for new malware families as well.

As the real-world distribution of malicious and benign macro ratios is not balanced, it is crucial to analyze the detection accuracy using an imbalanced dataset. This was addressed in [39] by selecting the most frequent words used in each class of Macros (Benign and Malware). Next, the performance of two different NLP feature extraction methods such as DocToVec and LSI, are compared and applied with four classifiers, such as MLP, SVM, CNN, and Random Forest. The results determined that using the proposed mitigation technique, the best combination of classifier and language model is SVM with DocToVec, which outperforms the related work performances. Furthermore, their model is less time-consuming than previous methods, mainly because of the mitigation technique applied in the pre-processing step.

Attackers frequently would obfuscate their malicious code to avoid being detected by Anti Viruses. Obfuscation is a very common practice for attackers to hide the logic of their malicious code. This can be achieved by selecting meaningless variables or function names or totally manipulating the program's control flow without affecting the final output, which will be further explained in the following sections.

Although obfuscated code is not necessarily malicious, there is a statistically high correlation between the two, making some research works purely focused on detecting obfuscated Macro programs. [30] divided obfuscation techniques in VBA code into four categories: Random obfuscation, Split obfuscation, Encoding obfuscation and Logic obfuscation. Moreover, a new set of features, named V features, are introduced in this work, and for evaluation, another feature set was built from the previous studies called J features. Finally, they compared the performance of 5 different machine learning algorithms using both feature sets. In general, they could achieve

better results using V features, with the MLP classifier performing the best.

As VBA source codes are sometimes not accessible, p-code is used to classify malicious macros in the document. One such work is done in [27], where p-code from different document files are used for the malicious macro detection. For this work, benign samples were collected from various repositories such as VirusTotal, Contextures, Fhvzelm, FreeVBCode, and Bearden & Lo, and malicious MSOffice samples from Canadian Cyber Incident were disassembled to p-code from the labeled VBA macros. The p-code extracted were then relabeled as malicious and benign based on the original document from where it is extracted, and a duplicate removal filtering process was done. The resulting dataset thus created was used as the experimental dataset, and the features were selected using the TF-IDF feature selection method along with Document Frequency and Fischer Score. In addition to that, the authors proposed a new set of features, and results showed that the proposed model performed better than existing models.

Lastly, it is important to note that macros can be manually enabled or disabled by the user, as mentioned in chapter 1. According to [26], it is best practice not to allow the macro for an attached office document from an unknown sender.

In this section, we can categorize document-based malware based on the common document types which are used for malware delivery, including:

1. Doc/Docx: Docx is one of the top file formats among the Microsoft word documents. Although document files of doc/docx format mostly looks clean, various multi-stage attacks can be triggered using these documents. A structural feature extraction methodology (SFEM) that outperforms the existing anti-virus engine is proposed in [19] for XML-based Office documents. In this work, the documents are recursively traversed for identifying the paths which are considered as features. For this, an imbalanced dataset of 830 malicious files and 16,108 benign files were applied with SFEM, and 131,747 paths were extracted

.Among the identified paths, unique ones were selected, and the frequency was noted. For dimensionality reduction and identifying the top features, feature selection methods such as Information Gain and Fisher Score were selected, and for feature representation, Binary and Term Frequency –Inverse Document Frequency (TFIDF) was considered. Based on the prominence, 20 CSV datasets were created and applied with nine machine learning algorithms using the WEKA tool. Moreover, authors conducted different experiments, and results proved that Random Forest performed better than other classifiers with Information Gain as the best feature selection method and Binary as the best feature representation method. This work also proves that the time complexity is reduced when numbers are removed from the features, and the model can be faster than the existing 2-gram and 3-gram methods.

2. PPTX: PPTX is another document file format that displays various content such as music, graphics, videos, text, etc. Sometimes, malicious PowerPoints are sent through emails where the malware can infect the system when the user hovers above the link. This type of attack made the researchers find solutions for protecting the documents. One such malware resilient file system is proposed in [38] based on access control mechanisms where the decision to access the document is determined. This work also rolls back the changes of the document if the temporary access given is suspicious. Furthermore, the model helps in preserving the integrity of file content. Results proved that the proposed model is efficient in building malware-resilient systems.

3. XLSX: XlSX is a spreadsheet file that stores data in compressed XML files. As most of the documents, xlsx documents are also attacked by malware. One such attack is the phishing campaign made by Evil corp where the xlsx documents are used for malware delivery [6]. As a solution for these malware attacks, different works such as [19] [38] were proposed.

4. RTF: RTF, also known as Rich Text Format is a document file format which is readable in several platforms. Even though macros are not available in RTF, malware can be embedded inside the document.In [49],a Vigenere-based obfuscation method, Kandi, is proposed where embedded malware is automatically uncovered by performing three stages: Plaintext extraction, Key length estimation, and Breaking the Obfuscation. First, the most frequent binary strings in the PE files are extracted by using a suffix array and the frequency count, followed by the Kasiski examination used for key length estimation. Finally, a probable plaintext attack is made against Vigenere-based obfuscation and results found that the proposed method can break the obfuscations and can expose obfuscated malware without parsing.

5. PDF: PDF, also known as a portable document format, is one of the widely used document formats that help make the document easily printed and non-editable. Therefore, the malware attacks in PDF are not easily recognizable. [45] explains the existing PDF malware detection approaches based on the type of analysis and automated detection approaches. For the type of analysis, the detection approaches were classified into static analysis and dynamic analysis, whereas for automated detection approaches, classification was done based on signature analysis, pattern matching, javascript features, metadata, and structural features. In addition to this, MS office malware was also investigated in this work with different detection approaches similar to that of PDF malware detection.

2.2 PDF Malware

Since PDFs are counted as a Page Description Language, it follows a specific structure with four components, illustrated in Figure 2.1:

1. Header: It is usually less than half a line that appears at the beginning of the PDF file and follows a specific format. This is the component that specifies the PDF version number.

2. Body: This is the largest and most important section of all PDF files, which represents all the content inside the document in terms of *objects*. Each object in a PDF has an index number followed by its type, such as Page, Font, Image Catalog, etc. In addition, objects can reference other objects that may have parents or kids. Hence, the PDF object structure resembles a tree, and once the root object is loaded, it loads the rest of the internal objects.

3. Xref: This is a table that allows us to have random access to PDF objects and stores each object's positional information.

4. Trailer: It specifies the root object among objects in the body part and specifies the location of the Xref table.

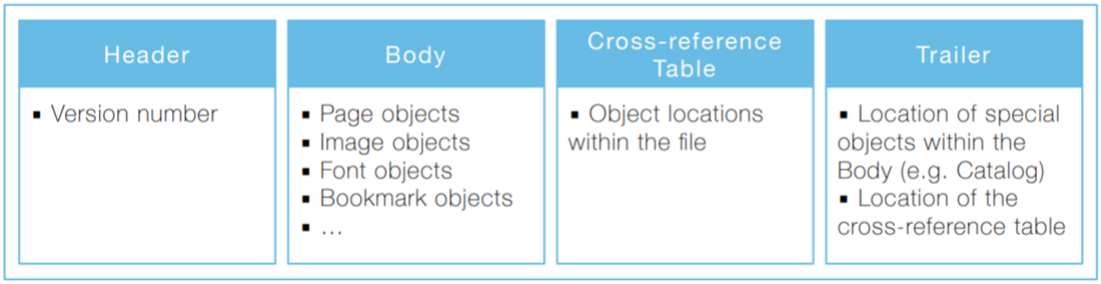


Figure 2.1: PDF Internal Structure

We can observe the PDF internal structure by opening a PDF file in a simple text editor such as notepad, and it will appear as shown in Figure 2.2, which is an example of a typical PDF body. As shown in the figure, obj four is not readable inside a text editor and these objects are called stream objects which stores binary data. Objects containing streams are usually filtered or encrypted using specific

compression methods supported in PDF. In this figure, the stream is filtered using "FlateDecode" as specified in the object dictionary.



Figure 2.2: PDF Internal Structure/Encrypted Object

2.2.1 Exploiting PDF Features For Delivering Malware

In this section, we take a closer look at PDF specifications and features and provide more details on how they can be exploited for malicious purposes. As specified in [16], PDF features can be exploited in different ways. Inspired by a couple of references [34, 16], we provide some more detail about what researchers generally look for in each feature to detect a malicious PDF.

1. Headers: When Anti Virus scans PDF, Headers are the first part that appears in PDF files and are inspected.[34] indicates that if the header is obfuscated and does not follow the specified pattern, it is usually a sign of malicious activity as a way to evade the anti-virus scan detection.

2. Metadata: Metadata is an object type that contains information about the file, such as creating data or description. One of the techniques that attackers have adapted over time is to hide snippets of code within document variables normally used to describe the PDF itself, such as the document's title or subject.

3. Javascript: There are numerous ways that attackers use and inject javascript to deliver malware. For example, they use a Javascript code to connect to a malicious URL and drop malware, or they embed a shellcode in the Javascript, which is somewhere inside the PDF. That shellcode may exploit a vulnerability of the viewing software such as memory overflow or exploit the system itself.

4. Stream Objects: Malware authors usually hide the malicious Javascript code inside the stream objects since it has no length limitation.

5. OpenAction/AA object type (Launch/GoTo/URI/JavaScript): This object tag is risky as it specifies an action or script that gets automatically executed once the file is opened.

6. Action class, whose elements are operating actions on user's own action (e.g., activating a hypertext link). However, it is possible to fool and trap the user by using hypertext links, invisible forms, which are activated once the mouse cursor enters the invisible field.

7. The GoTo function performs and manages the placement within the active document, towards a specified destination. A valid destination can be a predetermined page or an object such as graphics, hypertext link, annotation, etc.

8. Embedded Files: As PDFs can attach different file formats such as doc, XML, EXE, etc., attacker may use this technique to bind a malicious to the PDF, which gets executed upon viewing the PDF. Although it is not allowed to

directly launch an EXE or bat file in an adobe DC reader, malware authors can get around this. Sometimes attackers attach embedded doc files like word or excel to the PDF as an alternative for EXE files where these files can contain malicious Macros.

9. RichMedia: It is used to embed media files and flash objects in PDF files, enabling attackers to attach a malicious media file to the PDF.

10. Encoding filters: Encoding or compression filters are essentially used in PDFs to reduce the size or hide some sensitive content. Various types of filters are used to encrypt PDF objects, such as Flatedecode, CCITTFaxDecode, A85, DCTDecode, JPXDecode, JBig2Decode, ASCII85decode, LZWDecode, and RunlenghtDecode. Malware authors may use less common filters as they are harder to decode to hide their malicious payload or apply multiple filters.

2.2.2 PDF Obfuscation

PDF obfuscation is widespread among attackers to obscure the PDF content, structure, or malicious embedded Javascript logic. Obfuscation practices can be divided into two categories:

1. Name Obfuscation: One common specific obfuscation used in PDF file is "Name Obfuscation". Sometimes, the names and strings inside the PDF are obfuscated, representing them in another form. This way, they may intend to evade the AV signature detection. For example, PDF supports Hexadecimal Encoding, which enables representing a JS tag as "J#53" or "URI" tag as "#55RI". There are several Encoding techniques that PDF supports. Namely, Hexadecimal Encoding, Newline escaping. Octal encoding, Hexadecimal whitespacing and Encryption [46].

2. Embedded Javascript Obfuscation: As Javascript is the most common attack surface for PDF files, attackers tend to obfuscate it using various methods. Hence, it is important to look for code obfuscation signs in the PDF as potential malicious intent [2].

2.2.3 PDF Malware Detection

Analyzing the existing works on PDF malware detection is a plausible solution for proposing a novel PDF malware detection model. On this basis, various related works were selected from existing surveys as well. One such work is [25] that addresses several PDF-related attack scenarios such as OpenAction feature, launch action, embedded files, GotoEmbedded action, and URI action. Furthermore, the work analyses different PDF malware detection mechanisms based on static analysis, dynamic analysis, hardware, and machine learning. Finally, the work presented the perspectives on the study and defenses for evasion attacks.

[36] investigates PDF-based attacks by carrying out the study in different categories such as attack types, PDF detectors, and system evasion. For attack types, JavaScript-based Attacks, ActionScript-based Attacks, and File Embedding techniques were explained by providing a brief idea of the exploited vulnerabilities. For PDF detectors, existing machine-learning-based approaches were studied based on three stages such as pre-processing, feature extraction, and classifier. In addition to that, various parsers such as PDFId, Origami, Poppler, and different attacks against malware detectors like adversarial attacks, evasion attacks, poisoning, and privacy attacks were also examined in this study. The authors could also identify existing challenges and future research directions based on the analysis, thus helping in solving the challenges.

PDF malware detection techniques can mainly fall into two approaches known as static or dynamic, with some research that combines both methods. In the static

approach, the PDF file is analyzed without actually being executed whereas in the second one, an execution or simulation is required. Both of the approaches are discussed in more detail in the below paragraphs.

2.2.3.1 Static Analysis

In general, static analysis is about extracting static feature from a source code or a file and processing those features to predict whether it's malicious. It is explored in several studies and applied in many of the proposed techniques to detect malware. In the document-based malware detection area, static analysis is the most prevalent approach, which offers multiple advantages such as a faster detection rate, cheaper implementation cost, simplicity and ease of deployment. Static analysis can be further divided into three subcategories.

1. The first is pure signature-based methods to detect PDF malware, which most anti-viruses rely on for malware detection. This method is the easiest to implement but the most inefficient compared to the rest of the static detection methods.

2. The second being machine learning in which, the relevant features are extracted from the PDF and then fed into the machine learning models for classification. [42] has compared the five most relevant existing research works in the area of static malware detection where each research extracted particular static features such as N-gram, byte sequence, opcode sequence, etc. The paper analyzed these techniques in terms of advantages, drawbacks, and application. Moreover, an experimental study is also conducted where these features are fed to machine learning classifiers such as Decision tree and Roc -SVM, and results have been evaluated in terms of true positive and false positive metrics. It was observed that applying the N-gram method and the decision tree classifier resulted in 98 percent true positive outcomes.

3. The third static approach focuses on applying deep learning technology to classify PDF malware which appeared in relatively fewer research works, but gaining popularity. In 2020,[24] proposed a model that uses CNN to detect PDF malware. In this work, two models are designed where the first model is simpler while the second one is more complex but more robust. Both models perform almost on the same level as the best commercial Anti-viruses with no domain knowledge or feature extraction required for their development.

Static Features

The static PDF malware analysis is usually about extracting structural or metadata features from the PDF files to determine whether they are malicious or benign. We can divide these features into three main groups[43] [32]:

1. General Features: These features generally describe the PDF file, such as its size, whether it contains text or images, number of pages, and the title. Extracting these features usually does not require specific tools and can be done by manual analysis of the document.

2. Metadata or Structural Features: These features describe the PDF file in terms of the structure, which requires a deeper parsing. Features such as the number of objects within the PDF, object positions, average stream size, number of filters, types of filters, and pages are typical examples. Structural features provide an insight into the overall skeleton of the PDF; however, not much information is revealed about the actual content. Likewise, PDF metadata features that can be found partially in the properties section of the PDF contains information such as PDF size, creation date, author, and location.

3. Content Features: Another set of static features that can be extracted from the PDF files are content features. As the name suggests, these features represent the PDF content, which refers to what is inside the objects. These features

reveal more information about the document and can lead to more reliable predictions. Unlike structural features, extracting content features are not as easy. The main reason for this difficulty is, as mentioned in section 2.1, most objects tend to be encrypted using compression filters. To see what's inside the PDF objects, the applied filter has to be decoded first. This can be both time-consuming and complicated at the same time, particularly when filters are nested or less common filters are used.

In 2011, [21] defined 22 features that generally describe a PDF file, and developed an instrumented version of the MuPDF tool to extract them. To evaluate the effectiveness of the features, a machine learning classifier is trained. According to the results, the features such as "JavaScript in OpenAction" and "JS in OpenAction" stood out as a clear indicator of malicious intent. However, current PDF malware files generally use more sophisticated features for going undetected, which must be the future direction of this work. Likewise, in 2018, [50] extracted 48 structural and metadata features using off the shelf PDF parsers, and then they applied an MLP model to classify PDF files. Their model outperformed eight major AV scanners with 95% TPR and a low FPR of 0.08.

In 2015, [7] extracted features from 3 categories, general PDF structure, the structure of the objects, and seven features related to the PDF content. As a classifier, they selected the Adaboost algorithm, which consists of low-complexity weak learners and the experimentation proved that the proposed system outperformed the most recent TP, FP, and MCC approaches.

It is usually difficult to determine which learning system may perform the best and due to this reason, another work [47] compared the performance of three potential classifiers such a SVM, Random Forest, and MLP for PDF file binary classification. In this, 28 features that were a combination of features from the PDF metadata, document structure, and content are extracted in which a large portion of the features

was related to the embedded Javascript in the PDF. Then the PCA method was applied for feature reduction and a small training set from various sources is used and tested the model with real and recent samples. According to the test results, RF and MLP performed significantly better than SVM. However, one disadvantage of their detection system is that it parses the document, which implies complex operations, increasing the mean processing time for samples.

Static Analysis Challenges

One of the main challenges of static analysis is detecting and handling obfuscation. In 2016 [31], proposed a static technique to detect lexical obfuscation in Java malware which is quite prevalent. They extracted 11 indicative features from Java code according to statistical analysis and took advantage of cryptoanalysis techniques. These features were then fed to four machine learning classifiers where the random forest classifier performed the best at 0.99. Although Java is not used in PDF's, these features can be generic enough to be used for detecting javascript obfuscation in PDFs as well.

Another important challenge associated with PDF static detection using learning systems is that they are prone to adversarial attacks, and targeted malware PDF files may evade many of the structural and general features. Selecting stronger and more significant features is one of the strategies that helps tackling this problem. In [36], a solution is proposed which is specifically robust against evasive attacks by selecting the Adaptive Boosting method to classify the PDF files which trains a set of weak classifiers to build a stronger one. Using such learning system would make it difficult for attackers to evade the features as each weak learner is trained on a different feature set. Similarly, [22], proposes SVM as the learning algorithm, and different evasion attack techniques, specifically, the gradient descent attack were studied. Furthermore, they experiment the model with varying counter measures towards them, such as adjusting the threshold values for features. As a result, they

could completely block the naive evasion attack by setting a feature threshold value.

Static Analysis Tools

There are various open-source and commercial tools used for extracting static features from the PDF files. Many researchers employ them in order to extract their desired features. These tools offer their own advantages and disadvantages and may be different in terms of reliability. Some of the most common ones are listed below.

1. PDF examiner: An online virus scanner which analyzes the PDF files for javascript obfuscation.

2. PDFStreamDumper: This is a tool that analyzes each PDF object, decodes the filters, and most importantly, signifies potential exploits of the PDF file.

3. PDFId.py: This is an open-source python-based tool which scans a file to search for certain PDF keywords, allowing the user to identify PDF documents that contain suspicious keywords such as JavaScript or OpenAction. PDFid also handles name obfuscations, however, it is not able to look inside the PDF objects or decode them.

4. PDF parser: This tool can inspect PDF objects and display their content which also decodes some of the encoded objects with certain filters.

5. make-PDF.py: This tool enables the user to create PDFs with arbitrary Javascript code, and it enables them to embed files like doc, PDF, etc.

2.2.3.2 Dynamic Analysis

Besides static analysis, dynamic analysis is another technique that observes the code's behavior during run time instead of statically analyzing them. Adobe reader has used a sandboxed environment for this purpose where it runs the PDF in the limited locked-down environment to detect any suspicious behavior. Dynamic analysis is generally more reliable than static, and it's usually harder to be evaded by

attackers. It is specifically useful to detect the malicious javascript code embedded inside the PDF file since malicious javascript code is usually obfuscated, making static detection less effective.

One common technique to use the benefits of dynamic analysis without getting affected by their limitations is to combine both dynamic and static analysis to take advantage of both techniques. In 2014, [4] defined a set of both static and dynamic features indicative of malicious activity. To classify a PDF file, they first look for five static features and extracted the PDF javascript using document instrumentation and observe the execution of another set of dynamic features such as memory consumption and API calls. Their run time detector incurred a negligible delay of 0.093 seconds for each javascript code. However, one of the drawbacks of this work is that they focus on detecting malicious javascript exploits, whereas PDF files may exploit features other than javascript to launch malware.

Dynamic Features

Dynamic PDF features are the ones that reveal themselves during run-time. Features such as API calls, network communications, and suspicious memory consumption, which is an indication of heap spraying [34], are primary examples of such.

One of the works [20] that used a machine learning approach for malicious JavaScript code detection has three stages: API reference extraction, selection, and classification. First, the collected PDF files from various repositories such as VirusTotal, Malware don't need Coffee, Contagio, and search engines like yahoo and google were passed through the API reference extraction module that extracts the API references using static and dynamic analysis. Second, the suspicious samples were selected based on the threshold value and then used to create a reference pattern. Finally, the created pattern is classified into malware or legitimate and passed to a profound analysis if it is malicious.

Dynamic Analysis Challenges

Despite the more through inspection that Dynamic Analysis provides, it is not necessarily a better solution than static analysis and has a number of drawbacks. One important downside of dynamic analysis is that it usually incurs additional cost and time overhead for detection. According to [34], it also leads to additional complexity due to its execution in an isolated environment and tracking of the executable sections of the document like Javascript in PDF files. For example, attackers can hide a shellcode at weird places in a document, e.g., in the title, and reference it in forms like "this.info.title". In this case, the extracted javascript will fail to execute in emulated environments.

Most importantly, with recent attack sophistication, hackers can employ techniques to bypass dynamic analysis, and they can be aware that their malware is executed in a VM sandbox environment by inspecting VM system configurations,[33], or using the common simple trick which is applying sleep time to delay the execution of their actual payload until after the sandbox inspection is completed [13].

Dynamic Analysis Tools

1. Falcon Sandbox: Falcon Sandbox is a dynamic malware analysis tool that helps analyze unknown threats deeply so that security personnel can strengthen their defenses [3]. As PDF is also an attack surface, this sandbox has been upgraded with new capabilities for analyzing PDF files [8]. Due to its strong analyzing power, various malware analysis websites consider Falcon sandbox as an antivirus detection engine [7] [17].

2. Cuckoo Sandbox: Cuckoo Sandbox is another dynamic malware analysis tool [28] that helps in identifying the behavior of the suspicious file by executing them in an isolated environment [4]. This sandbox is used in PDF malware [5] and different categories such as windows malware [23], android malware [41], IoT malware [35], and Linux malware [48].

2.3 Phishing Attack Using PDF

As discussed in the chapter 1, PDF malware is either an exploit or a phishing attack. In this work, we are specifically targeting the exploit attacks. However, attackers frequently apply phishing and integrate it with other techniques to deliver malware, specifically document-based malware. Therefore, the significance of phishing techniques can not be ignored. In this section, we analyze standard phishing techniques, their properties, and their countermeasures.

2.3.1 Introduction To Phishing Attack

According to Wikipedia (https://en.wikipedia.org/wiki/Phishing), phishing is a social engineering attack that targets users to reveal their sensitive information or deploy malware on their systems where the user's weakness is exploited instead of their machine. As phishing can be launched through a website, email, or text message, it is generally integrated with heavy social engineering techniques to better convince the user into doing what the attackers want.

Nowadays, attackers employ very sophisticated and powerful phishing techniques that make many users fall victim. According to [9], phishing is the third most prevalent Cyber Attack resulting in information breaches.

Below we give an overview of the common phishing signs and techniques :

1. Redirection To a Phishing Website: Phishing websites are fake websites with a spoofed domain or with the same appearance as another legitimate website that the user intended to be on. For example, a phishing website may look just like a bank website and asks for the user's bank credentials and uses them for malicious purposes. Tricking the user into opening malicious website URLs can be done in many different ways. Sometimes the attacker uses a URL that looks very similar to the one of a legitimate website and this is achieved by

a subtle misspelling of the original URL or using special characters that look very similar to an English character. This typosqatting makes the user believe that there is no harm in clicking on the link. Sometimes the URL might be simply hidden, so the user has no idea if they click on that.

2. Phishing Malware Download: One of the common types of phishing practice is where the attacker tries to convince the user to download malware on their own system. This is usually carried out by attaching malware in an email and tricking the user into opening or downloading it on their system.

3. Requesting A Response Containing Sensitive Information: As most of the phishing activities are through emails, sometimes attackers try to deceive the user into replying to them by including their personal information, or just simply enticing the user to leave a general reply which could reveal further details for the attacker, such as the user activity status or their email signature for them to plan a more targeted attack later on.

2.3.2 PDF-Based Phishing

As mentioned earlier, one of the common ways of phishing is attaching malware and enticing the user into downloading it. However, according to [11], including malicious links or key spam words inside the email itself might lead to it being detected and blocked by plenty of email gateways, whereas attaching a document would evade this detection easier, since scanners usually do not scan inside the attached documents. Receiving phishing PDF files is quite common among users since most of the shared documents in emails have PDF format, and it is not blocked by the email server, as opposed to other file types such as .exe or .bat. Moreover, PDFs have powerful features such as 3D image support and Javascript effects that attackers can leverage to deceive the users more effectively than a simple text-based email.

To create a phishing PDF, the attacker essentially includes all the malicious content in a typical phishing email inside a PDF file attachment and the email text is written to trigger the user into opening the attached document.
Here we present some examples of the tricks that attackers employ in a phishing PDF file.

1. They use malicious URLs in the format of a legitimate button and trick the user into clicking it, as illustrated in Figure 2.3, sometimes they put a static luring image with a crafted play button which takes the user to an exploited URL or it might directly download malware on the user's system, as depicted in Figure 2.4.

2. They use well-designed PDF forms, and ask for sensitive information from the user and send them to a malicious website or to the attacker upon clicking on the submit button.

2.3.3 Defending Against Phishing Attacks

There are various techniques and tools developed to detect and prevent different types of phishing attacks. Although phishing emails in particular can sometimes be manually detectable by the user, some are much harder to detect. Many phishing emails spoof the address of other trusted emails, which is usually successfully detected by the SPF technology developed by [12].
However, sometimes phishing emails do not spoof another email address, and they mainly focus on delivering a clever, phishy content for scamming the users. This makes most of the blacklisting and whitelisting methods ineffective. Hence, analysing the content is crucial to detecting suspicious and phishing activities.
Content-based phishing analysis is in fact a type of text-based analysis, which identifies phishing activities based on the given content. It is applied in numerous email

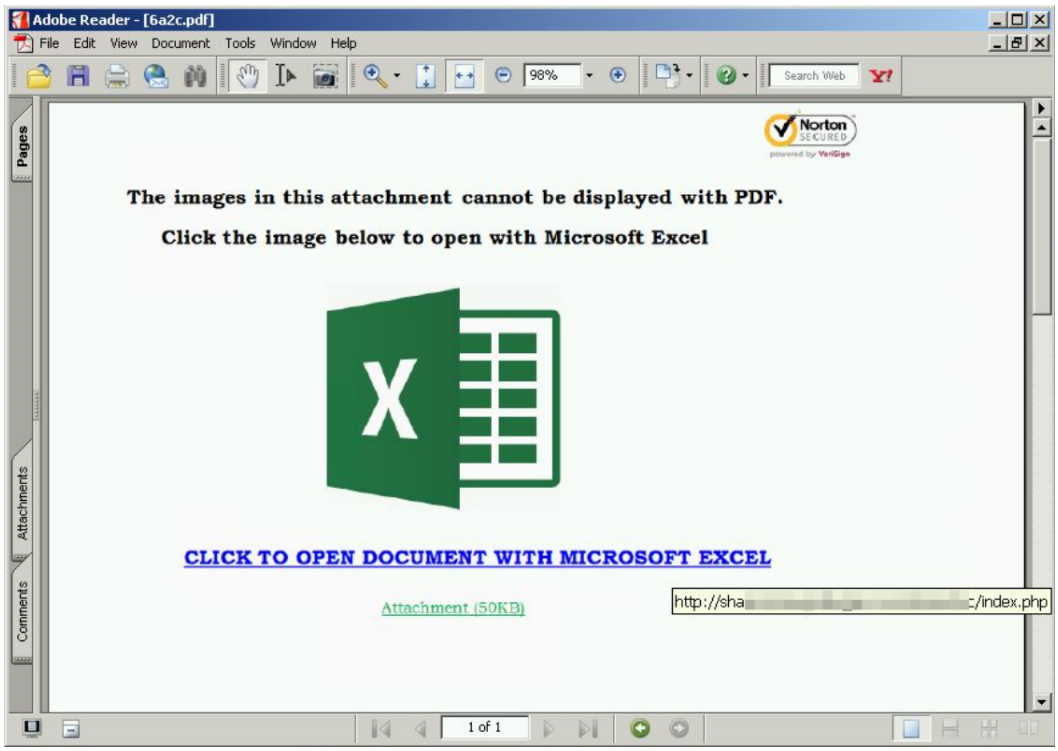


Figure 2.3: PDF Phishing With a Hidden Malicious URL Taken From [10]



Figure 2.4: PDF Phishing With Static Image Resembling a Video Taken From [10]

spam detection techniques in which the email content is taken as an input and is flagged as a spam based on content features such as the presence of certain keywords or images. If we specifically target PDF-based phishing, all the content-based phishing detection methods can be applied to detect scamming PDF files.

Many studies incorporate AI technologies such as machine learning to detect spamming and phishing. [44] extracts 58 features such as the existence of forms, number of buttons, number of hidden elements, and website title from the websites' content. The features are then fed to eight different machine learning algorithms to classify the websites and results showed that Xgboost performed the best at 97.88%.

Phishing contents usually attempt to convince users to take a certain action whether it is downloading a file, opening a URL, clicking on a button, etc. Incorporating this fact, [18] constructs a semantic web to translate extracted keywords from a text to an event and paired every two events with close semantics and represent each email content as those pairs of events. Finally,results were fed into a "Fuzzy Control" which is an artificial intelligence technique to classify them. Their technique discovers a relevance ratio between events and categories to judge whether the email is legitimate or phishing.

2.4 Comparing PDF Malware Detection Techniques

In this section, we provide a comparative analysis of the common PDF malware detection approaches. Based on this study, we could identify that majority of the research employs static methods and machine learning to detect malicious PDF files, and SVM is the most popular machine learning algorithm.

1. No Additional Resource Required: Static analysis does not require a separate emulator or protected environment for running the program. This saves additional costs.

2. Less Time Consumption: Dynamic analysis generally incurs additional overhead for running the program instead of the static analysis. Given that Parsing the PDF to extract features is already a quite time-consuming task, it is important to minimize the processing and detection time overhead as much as possible.

3. More Reliability And Simplicity Of Deployment: Although static analysis and static features can be evaded, but there are countermeasures available to reduce this risk, such as selecting stronger and more significant features or using thresholds for the features. On the other hand, dynamic analysis requires executing the whole PDF file, which is not easy for various reasons mentioned in the previous section. At the same time, it doesn't necessarily offer less possibility of evasion as dynamic analysis can also be evaded with various tricks. For example, hackers can be aware that their malware is run in a VM sandbox environment by inspecting VM system configurations [33].

However, there are certain aspects in which dynamic analysis is better than static analysis. It includes:

1. Identification of vulnerabilities that are not detected during static analysis

2. Less false positive rate as the runtime behaviour is considered.

2.4.1 Machine Learning Solutions Vs. Deep Learning Solutions

There are relatively more researchers focusing on Machine learning solutions for PDF malware detection, which is further presented in Table 2.5. One important reason could be associated with their interoperability and scalability. In addition to that, machine learning techniques require fewer training resources, and the training speed

is usually much higher than deep learning models. Machine learning solutions can also learn from much less amount of data.

On the other hand, Deep learning models typically require much larger datasets to converge to a solution and usually offer higher accuracy. Also, these models are generally more reliable than machine learning models as they can extract super features on their own, which in turn results in higher accuracy. Deep learning techniques also automatically handle a concerning issue, which is feature extraction. As manual feature extraction for PDF files suffers from various issues specifically outlined in 2.2.3.1, deep learning based approaches can be a powerful solution for text/binary file analysis which requires extracting a large number of features.

2.5 Remaining Concerns

As noted earlier, PDF is a very flexible file type that supports a lot of features. For example, the objects in the PDF file can appear in any order and refer other objects with no limitations, and some objects can nest other objects inside them. Various hard-to-decode compression filters and Obfuscations are supported in PDF. Hence, all these features and flexibilities offered by PDF files make obfuscations and evasions even easier for attackers. This is why extracting features specifically indicative of obfuscation is very important.

This PDF complexity also leads to another issue which is the feature extraction reliability. When we want to analyze a PDF file, whether manually or through any learning system, we need to extract the corresponding features in the first place. Although there are many tools available for this purpose, none of them can be considered 100% reliable because of the aforementioned issues.

Classification speed is another notable concern in PDF malware detection. The complicated structure of PDF files and all the evasion possibilities that PDF format

offers make PDF parsing a time-consuming task, which affects the classification processing time. Therefore, we must consider countermeasures such as selecting fast-learning systems, reducing the number of features, and optimizing feature extraction techniques to minimize this issue. Table 2.5 is a comparison of the main relevant works and their proposed solutions in the PDF malware detection area.

Table 2.1: Relevant Research Work Comparison

Ref	Method	Solution	Advantage	Disadvantage
[34]	Dynamic Analysis	Hybrid (static and dynamic)	Fast and robust against evasion because of the dynamic analysis	Cannot detect malicious PDFs that do not use Javascript
[50]	Static/ Deep Learning	Using MLP	Low FPR of 0.08	No countermeasure for evasion attacks, no justification in the feature or model selection, No detection speed evaluation, no comparison with other works
[24]	Static/ Deep Learning	Using CNN to detect malicious PDF	No feature extraction required while performing at the same level as common anti-viruses	No countermeasure for evasion attacks, No evaluation for detection speed, no comparison with other anti-virus performances,no other deep learning models are investigated

[47]	Static/ Machine Learning	28 final features are selected using PCA along with dynamically using three different ML classifiers (Support Vector Machine, Random Forest, and Multilayer Perceptron)	Harder to launch evasive attacks, as a comprehensive set of features are extracted from metadata, structure, and content. Time consumption considered for selecting the candidate models. Models are tested against real-life examples as well as relevant baseline datasets, which represents how generalized their model is.	The assumption is based on constantly retraining the model, which can be costly. Extracting and processing features from a heterogeneous set of inputs with a high level of granularity implies a higher processing time than other approaches.

[29]	Static/ Deep Learning	Two-layered CNN is proposed along with embedding	Sufficient theoretical justification of the proposed model and the structure. Minimal effort required for feature extraction or data processing. Training speed taken into account for the selection of the best model	Oversimplified and naive dataset labeling process, No explanation provided for the choice of data pre-processing (sequence length ,padding technique). *prediction is merely on the basis of streams.*
[36]	Static/ Machine Learning	Features selected from both the structure and the content of the PDF file, using Adaboost as a classifier	Inspected evasion attacks and tuned their proposal to tackle them. Defined significant features for extraction	Decision function is very dependant on the quality of the training data. Detect targeted and optimal attacks.

Based on our studies in this chapter, Figure 2.5 presents our proposed general taxonomy of PDF malware (Achieved *Con2*).

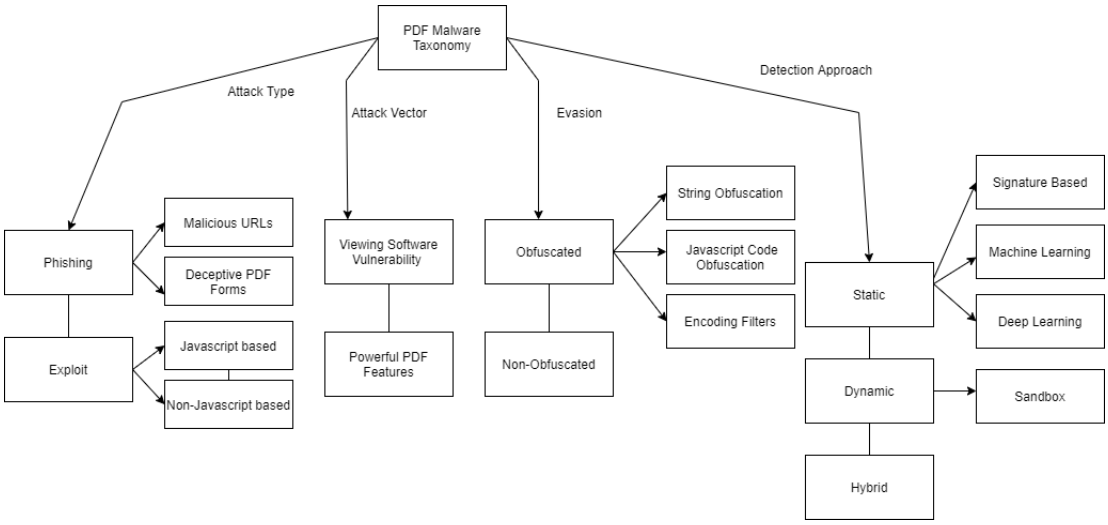


Figure 2.5: Taxonomy Of PDF Malware

2.6 Concluding Remarks

In this chapter, we provided key terms and concepts regarding PDF files and malware detection. Furthermore, we could observe how the complicated PDF structure provides a vast surface for attackers to misuse as shown in our taxonomy (Achieved *Con2*), and how challenging it is to extract the required features from PDF files for analysis. Finally, we could identify that both Static and Dynamic detection approaches have their own challenges and by comparing different proposed detection methods, we could see that newer research works tend to shift more and more towards tackling evasive attacks. In the next chapter, we are going to discuss the motivation behind this work and how our proposed approach can contribute to the improvement of the previously proposed techniques.

Chapter 3

Proposed Approach

By analyzing existing works, it is identifiable that static-based approaches are widely used for detecting malicious PDFs, and a majority of them tend to rely on a single machine learning or deep learning model. However, to the best of our knowledge, no research in the literature has detected evasive malicious PDF files by combining different classifiers for predicting the final result, which is a practical approach as identified in [51], that applied Stacking for spam email detection. Unlike a single classifier, stacking uses several classifiers that make unique assumptions on the data to predict the output, leading to better performance when combined in the suitable form. This idea motivated us to propose a detection solution for PDF malware detection based on stacking learning. In this work, 37 static representative features are extracted from each PDF file and fed into our proposed model for detection. We provide our testing and implementation in the following sections to demonstrate the efficacy of the proposed approach.

3.1 General Overview

Before we explain the details, we present the general layout of our proposed framework, which consists of two modules as illustrated in Figure 3.1. The components of

model include:

1. Raw PDF Files: The raw PDF files from the user is taken as an input and directly fed into the Feature Extraction Module.

2. Feature Extraction: In this module, relevant features are extracted and necessary pre-processing is applied which is explained in the following sections.

3. Detection: The extracted features from the previous step are then passed to our proposed detection model for classification.

4. Prediction Results: Once the the detection module has classified the PDF file, the final prediction is generated which is a binary label: malicious or benign for each PDF.

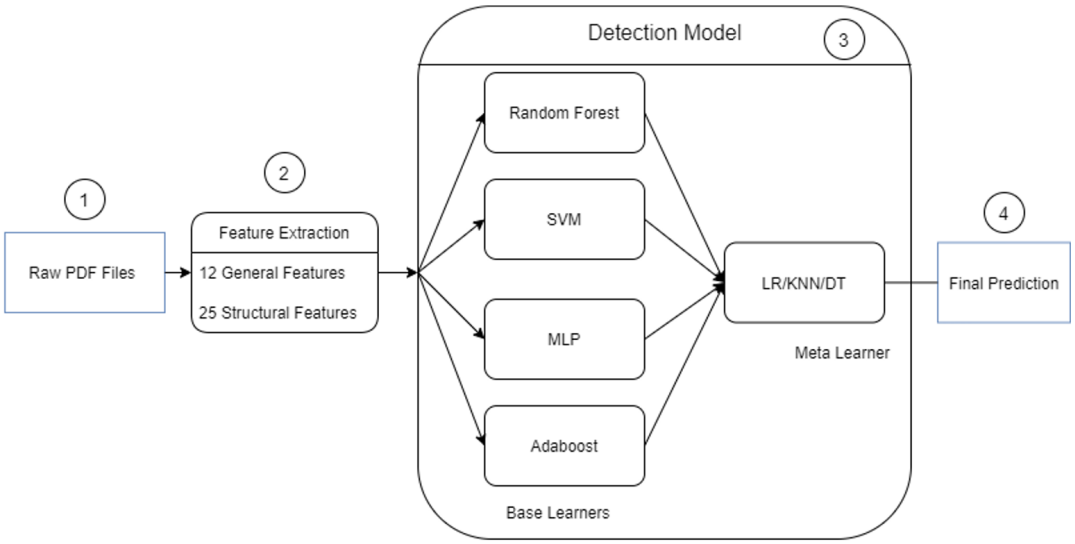


Figure 3.1: Layout of the proposed architecture

3.2 Feature Extraction

In order to classify each PDF, we propose the extraction of 37 features which are selected in a context aware approach based on multiple observations and studies.

Selected features are divided them into two groups of General and Structural as mentioned below.

- **General Features**: As discussed earlier in the literature review, these features are the ones that generally describe the PDF file. We selected 12 features from this category as listed below

    1. Number of characters in the title: Legitimate PDF files usually have a proper and more meaningful titles
    2. Metadata size: Metadata is the section where information about the PDF file is provided which can be exploited for embedding hidden contents.
    3. Document Encryption: This feature shows whether the PDF document is password protected or not.
    4. Number of pages: Malicious PDF files tend to have fewer pages(majority of them have 1 blank page) as they are not concerned about content presentation.
    5. Presence of text inside the PDF: As content presentation is not the objective of malware PDF files, they may include less text in their files.
    6. Size of the whole document: The malicious PDF size usually tends to vary from the benign,due to its variation in page size and content
    7. Number of embedded files inside the document: PDFs are capable of attaching/embedding different types of files within themselves that might be used for exploitation, including other PDF files, doc files , images , etc.
    8. Average size of all the embedded media: Embedded files in the PDF may be of various sizes depending on what they contain. The average size might lead to an insight on the embedded files content.

9. Number of total objects inside the PDF: As shown in chapter 2, PDFs are made of objects. Number of objects combined with the rest of the features can well represent the PDF in general.
10. Number of font objects: Font objects indicate the types of fonts that are used for the PDF text.
11. Presence of a valid PDF Header: As PDF header obfuscation is a common practice for evading anti-virus scan, malicious PDF files tend to modify the header format.
12. Number of images in the document: PDF files may contain one or any number of images. Malware PDFs can specifically misuse images in many ways, such as hiding a malicious code using techniques such as Steganography.

- **Structural Features**: We propose a set of 25 features that are related to the structure of the PDF, as specified below.

1. Number of indirect objects: This might be some indication of obfuscation attempt.
2. Number of obfuscations: PDFs support many types of obfuscations such as string obfuscations of hex, octal etc. which are generally applied for evasion attempts.
3. Number of streams: Shows the number of sequences of binary data in the PDF.
4. Number of of endstreams: Keywords that denote the end of the streams.
5. Average Stream size: Size of the stream as the malicious code maybe hidden inside streams.
6. Number of stream objects (ObjStm): Streams that contain other objects.

7. Number of Javascript keywords: This denotes the number of objects containing Javascript code, which is the most commonly exploited feature as obvious.
8. Number of JS keywords: Number of objects containing Javascript code.
9. Number of Launch keywords: Launch is a keyword that can be used to execute a command or program.
10. Number of URI keywords: Indicates a presence of URL which the PDF file attempts to connect to.
11. Number of Action keywords: Specifies a certain action upon an event.
12. Number of AA keywords: Specifies a certain action upon an event.
13. Number of OpenAction keywords: Specifies a certain action upon opening the PDF file. This feature combined with Javascript has been observed in majority of typical malicious PDF files.
14. Number of SubmitForm tags: Indicates the PDF button that collects form information and sends them to specified destinations.
15. Number of Acroform tags: Acrobat forms are PDF files that contain form field which support scripting technologies that can be misused for attackers.
16. Total number of filters used: There are various types compression filters applied on some PDF objects, which also might be exploited by attackers.
17. Presence of JBig2Decode filter: JBig2Decode is a common filter, to encode malicious content.
18. Number of objects with nested filters: Nested filters can be an indication of evasion, as they make the decoding process more difficult.
19. XFA: XFAs are XML Form Architecture that are included in certain PDF

files which support scripting technologies that can be misused for attackers.

20. Colors: Different colors used in the PDF.
21. Trailer: Number of trailers inside the PDF.
22. Xref: Number of Xref tables.
23. Startxref: Number of keywords with "startxref" which denotes where the Xref table is started.
24. Xref enteries: Number of entries in the PDF xref tables as malformed Xref tables are another common observation in malicious PDF files.
25. RichMedia: Number of RichMedia keywords which denotes the number of embedded media and flash files.

Once these features are extracted from each file, they will be input to our proposed detection model which is further explained in the next section.

3.3 Detection Model

As the main idea behind our approach is based on the Stacking method, we present a brief overview on the Stacking technique before we dive into discussing our own model.

3.3.1 Overview Of Stacking

Stacking at the simplest level can be defined as a method which arrives at the final output based on the outputs of a set of different classifiers. These classifiers are addressed as the base learners which learns the output from the original dataset. The base learner predictions will be input for a single or set of other classifiers called meta learners and these meta learners are supposed to learn from the base learner

predictions and produce the output. Base learners and Meta learners are chosen among any of the machine learning or deep learning classifiers, according to their fitness for the problem and other factors.

3.3.2 Training And Testing Stacking-Based Classifiers

Here is a step by step explanation on how we can predict an output, through Stacking.

1. First, the entire dataset is divided into K-Folds where K is set depending on the size of the dataset and other relevant factors.
2. All the base learners are then trained on the k-1 folds of the dataset, separately.
3. Each of the base learners are tested on the k-th fold of the dataset and then the outputs are all gathered in a single file.
4. Step 1 and 2 are repeated k times for each of the K-folds, until we have the classifiers predictions for the entire dataset.
5. The predictions from the base classifiers are fed to the meta learner/learners as predictors, and the actual dataset test label is input as the target label.
6. The meta learner is trained on the k-1 folds of the outputs and generates outputs on the k-th fold of the dataset.
7. Steps 5 and 6 are repeated until we reach to the end of the stacking level, and the generated outputs at that level, are the final results.

3.3.3 Proposed Stacking Architecture

In this section, we aim to define our proposed detection architecture which is based on stacking learning. In order to determine that, we need to specify several primary parameters first. Below, all the required parameters are discussed in further detail.

3.3.3.1 Number Of Stacking Levels

The first trivial variable that has to be set for Stacking is the number of stacking levels. Although adding more stacking levels helps learning at a deeper level, it incurs more complexity and cost of implementation. Therefore, it is important to arrive at a compromise between our desired test results and the cost. In this work, given the problem complexity, we preferred two stacking levels, which is a suitable trade off that could lead to a reasonable accuracy with a low cost.

3.3.3.2 Number Of Base Learners

Number of learners can vary from one to any number of learners and increasing or decreasing the number of base learners does not necessarily lead to an improved result. Hence, the number should be determined based on various factors such as the nature of the data, number of features, hardware cost and time consumption. In this work we selected four base learners, and we will discuss our rationale for their selection in the following.

3.3.3.3 Types Of Base Learners

Identifying the types of base learners is one of the most challenging and important steps for defining our model, and it has to be achieved in a careful manner for maximizing the benefits of Stacking. Before we discuss our proposed base learners, we outline some main criteria for selecting them.

Firstly it is important to note that selecting complex and strong classifiers alone is not helpful and does not necessarily achieve better results. The best practice is to select diverse classifiers in a way that each bring a different perspective of the data to the table, so that their predictions complement each other.

Another notable factor is to consider the nature of the problem and the dataset that we are dealing with.

Considering this, we have formulated the criteria as:

- Dataset with of 37 features.
- Majority of continuous feature values
- Type of classification is binary
- Correlated features

Considering the above-mentioned facts, we selected SVM, Random Forest, MLP and Adaboost as our potential base learners and further justification for choosing each of them is explained below.

1. Random Forest: Random Forest is a tree-based algorithm which has a high performance particularly with larger datasets with sufficient number of features. As the number of features are not large, we are going to use 10-50 tree estimators, which we aim to finalize in the experiment section.

2. Support Vector Machine (SVM): SVMs are suitable machine learning algorithms when dealing with a relatively large dataset, which is however large enough to be fed to deep learning models. Moreover, SVM has been a very popular and successful classifier with several text-based analysis tasks. SVM models use a certain functionality called as kernel, which essentially defines how the input should be mapped to the output. There exists different kernels used in SVM algorithms such as Polynomial, Linear and Gaussian Radial Basis. We particularly selected the linear kernel due to the following reasons:

   (a) Faster

   (b) Less prone to overfitting

   (c) Since we are having a small number of features compared to the dataset entries, Linear Kernel will most likely produce reasonable results

3. Multi Layered Perceptron(MLP): MLP is one of the most typical types of Neural Networks. Neural Networks are powerful models that employ a totally different logic than the other Machine Learning classifiers for prediction. This is why stacking one Neural Network with the rest of our classifiers, at the minimum can add a useful variety. We selected MLP, as it does not require a very large number of data samples to converge unlike CNN, and it handles non time series data, as opposed to RNN.

4. Adaboost: As the fourth potential base learner, we selected Adaptive Boosting (Adaboost) which is a tree-based algorithm, that works by training a set of weak learners and sequentially boosting them to build a stronger model. Moreover, Adaboost is a strong model which is less prone to overfitting.

3.3.3.4 Number Of Meta Learners

In stacking, only one classifier can be selected as a meta-learner as we already chosen two stacking levels, which helps building a more cost effective solution.

3.3.3.5 Types Of Meta Learners

Selecting the most suitable meta learner type depends on multiple variables. Similar to base learers, same key factors are considered for selecting the meta-learner as well. Below we listed the factors that we need to take into consideration for determining suitable meta learners.

- The dataset input to the meta learner consists of minimum of two and maximum of four features with one target variable. This depends on the finalized number of base estimators selected later through experimentation.

- The data size is similar to the original dataset as the data points that our meta-learner is trained on are generated by predictions of the four base learners that

we previously defined.

- Data features are categorical and can only take two possible values:malware or benign.
- Classification type is binary
- The learning problem is not complex as the highest number of features are four, and the target variable has only two categories.
- All data features are independent of each other as they are produced by different predictors.

Considering these aspects, three different classifiers are shortlisted as potentially fit Meta learners for our problem. However, the selection among three will be finalized later in the experimentation phase. Here, we briefly justify our choices for each of the three classifiers.

- Logistic Regression: This is the most common meta learner used in majority of stacking practices for binary classification. It is a quite fast algorithm that ignores the correlation between features and is best suited to work with categorical data. The only relevant challenge is that Logistic Regression attempts to linearly separate the data points, while it is not always possible to ensure whether our data is linearly separable or not. This is what we aim to explore further in the experimentation stage.
- K-Nearest Neighbor (KNN): It is a simple model, which usually provides a high accuracy and i tends to perform particularly well when dealing with many instances with a few dimensions which is similar to what we are working with to train and test the meta learners.

- Decision Tree: Decision Tree classifiers are fast predictors that require limited computational power while providing a reasonable accuracy. Moreover, they are a suitable choice for working with categorical features.

In this chapter we theoretically specified the proposed architecture as shown in the Figure 3.2, and the best combination of selected base learners for stacking is yet to be determined in our experimentation step.

3.3.3.6 Dataset Split

There are variety of ratios to use for train-test split of our dataset. For that, K-fold cross validate has to be done for each of the single classifiers before being able to train the stacking model. Choosing a large value for K leaves a very small portion of data for testing, which might be misrepresentation, if the dataset is not large enough. On the other hand, we have less training data and more testing data for smaller value of K. Considering this, we decided to set the K value to five, which is somewhat a trade-off between computational complexity, and accurate skill estimation of the model.

3.4 Handling The Overfitting Issue

Overfitting is a common issue that occurs when working with complex learning models. A strong indicator of overfitting is when the model performs very well on the training set, while performing poorly on the testing set. Unlike Ensemble models such as Bagging and Boosting that have in-built measures to prevent overfitting, Stacking is subject to this issue, particularly when the base learners are strong classifiers with a high variance. In this work, we plan to avoid overfitting as much as possible, by applying two simple countermeasures as below :

1. Selecting base models among the ones that are less likely to overfit.

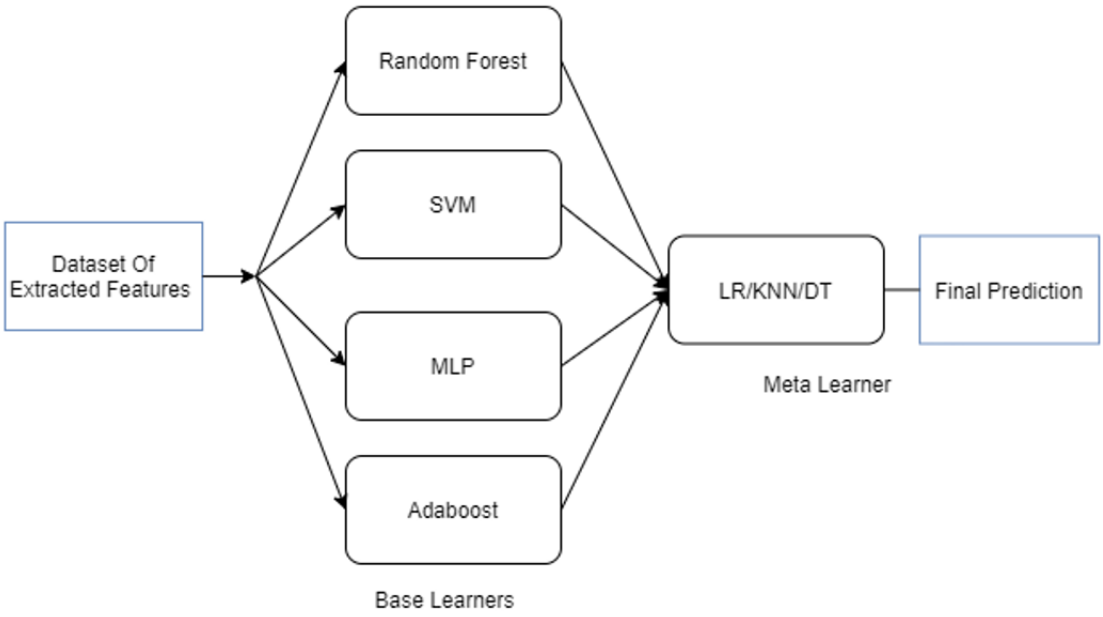


Figure 3.2: Proposed Detection Model

2. Tuning the base learner parameters in such a way to avoid over fitting.

Considering this, we tried to select our base predictors in a way that they are less likely to overfit. For example, Random Forest and Adaboost are intrinsically robust against overfitting. Linear SVM is also a simpler version of SVM that reduces the risk of overfitting. However, MLP is a complex model that generally suffers from higher chances of overfitting and does not handle it intrinsically. Therefore, here we apply the second countermeasure and tune the MLP parameters, in such a way that minimizes this risk.

There are generally two overfitting prevention approaches that can be applied to neural networks:

1. Early Stopping: In this method, a validation set is specified, and the training epochs are stopped as soon as there is no improvement observed on the validation dataset.

2. Regularized Cost Function: This method works by applying a certain amount

of penalty on the cost function. There are two main Regularization methods available.

(a) L1 method: In this method unimportant features are removed, which is useful when working with high dimensional data. Moreover, L1 method is robust to outliers.

(b) L2 method: This method is often reported to be more precise than L1 in majority of cases and is more cost efficient. However it is less robust to outliers, and it does not reduce the feature set.

We decided to follow the second approach to overcome overfitting as it is less dependant on the specific training data. Also, we particularly selected the L2 method as we prefer not to reduce our feature set since it is already relatively small and specific L2 rate is set in our implementation section (Achieved part of *Con1*).

3.5 Concluding Remarks

In this chapter, we presented our detection architecture in full detail. Moreover, we specified the set of our selected features and then defined our detection model structure and tuned the required variables, with the help of theoretical justification (Achieved part of *Con1*). In the next chapter, we will describe our implementation of the proposed architecture, and specify the tools and source codes that were used.

Chapter 4

Implementation

This chapter provides the implementation details of the framework presented in Chapter 3 along with the experimentation setup. In addition to that, we include the code along with the tools and libraries that are utilized as part of the implementation.

4.1 Overview

In order to give an overall picture of the implementation for our detection framework, a block diagram is provided in Figure 4.1, and each component is elaborated, and the relevant implementation details are explained in the following sections.

4.2 Experimental Setup

We implemented and executed all modules in a Kali Linux OS with one vCPU and 4 GB RAM on a Virtual Machine for the experimental setup. The components were developed using the Python programming language.

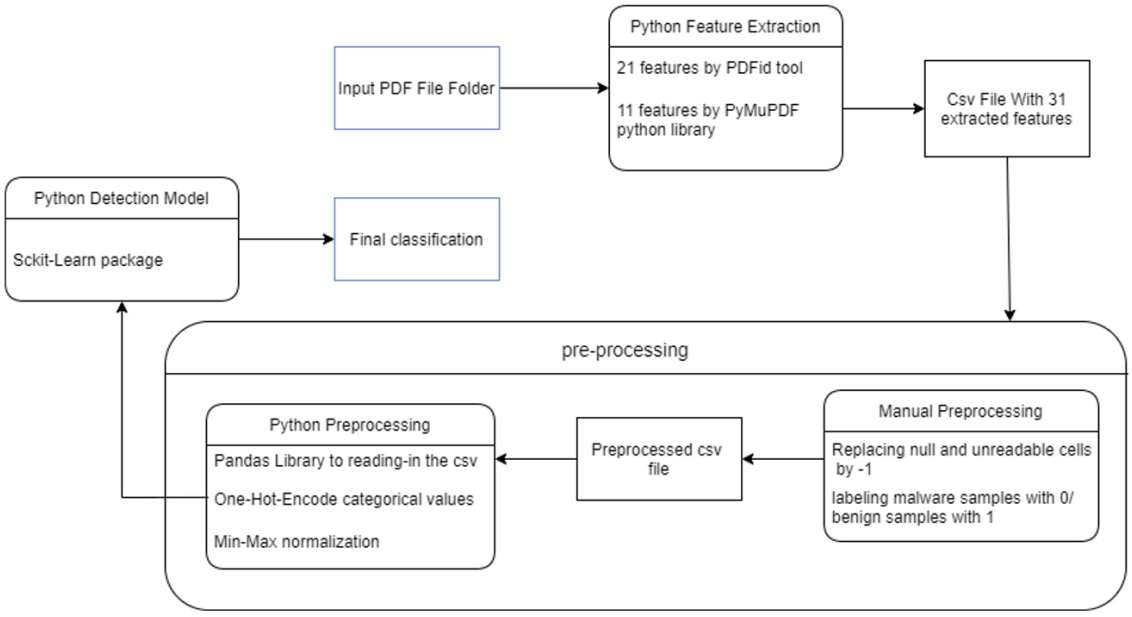


Figure 4.1: Implementation Overview

4.3 Feature Extraction

As explained in Section 3.2, our feature extraction consists of extracting two sets of features: general features and structural features. However,some of these features are removed due to extraction's inefficiency and complexity, which would incur a relatively high computational overhead on our system.

Below we listed five features that we decided to exclude from extraction for each PDF file:

1. Size of all the embedded media
2. Number of the string obfuscations
3. Average Stream size
4. Number of objects with nested filters
5. Number of font objects

Excluding the above features, 32 features were left, and for extracting them, we developed a python script that reads in a folder of PDF files and extracts 32 features from each and writes them on an output CSV file.

4.3.1 Tools And Libraries

In order to develop our python script which extracts different parts of our feature set, certain tools and libraries were used. It includes:

1. PDFid: As specified in section 2.2.3.1, PDFid is an open-source python tool with a CLI interface developed by Didier Stevens. It extracts certain features from a single PDF file and displays the feature values on the command line interface. In this work, this tool was used to extract 21 of our desired features. In order to achieve this, we made a few minor modifications to the actual source code. We developed a python script that executes this tool as a shell command to extract the features from a set of PDF files in a folder and write them on an output CSV file.

2. PyMuPDF: This tool is used to extract the remaining 11 features by importing the library into our code. Once this was done, all the extracted features were set with PDFid and moved onto a single CSV file.

4.3.2 Coding Details

The main coding snippets of our feature extraction script are depicted in the Figure 4.2 and Figure 4.3. Moreover, the entire developed source code is available in our Github repository https://github.com/ahlashkari/PDFMalLyzer/ (Achieved *Con5*).

```
for j in os.listdir(path):
        f = path + "/" + j
        print(f)
        try:
                doc = fitz.open(f)
        except:
                continue
        #metadata
        metadata = doc.metadata
        print("metadata is "+str(metadata))

        #title
        if metadata:
                title = metadata['title']
        else:
                title = ""
        if not title:
                title = ""
        print("title is "+str(title))

        #number of objects
        objects = doc.xrefLength()
        print("object is "+str(object))

        # printing number of pages in pdf file
        numPages = doc.pageCount
        print("numpages is "+str(numPages))
```

Figure 4.2: Code Snippet For Extracting Features Using PyMuPDF

```
print("extracting structural features...")
#extracting structural features using pdfid
var =  str(r"tr '\n' ','")
command = ""
header = ['header','obj','endobj','stream','endstrean','xref','trailer','startxref','pageno' ,
          'encrypt','ObjStm','JS','Javascript','AA','OpenAction','Acroform','JBIG2Decode',
          'RichMedia','launch','EmbeddedFile','XFA','Colors']
with open(os.path.relpath("pdfid/output.csv"),'w',encoding='UTF8') as output:
        output.write(','.join(header))
        os.chdir('pdfid')
        t0 = time.time()
        for j in os.listdir(path):
         f = path + "/" + j
         out = subprocess.getoutput("python pdfid.py "+f+" | awk '{print $2}'
                                                          | tail -n +2 | "+var+"")
         output.write("\n" + out)
         d = time.time() - t0
        print("duration: %.2f s." % d)
os.chdir('../')
print("finished features")
```

Figure 4.3: Code Snippet For Extracting PDFid Features

4.4 Pre-Processing

To make the CSV file clean and understandable, preprocessing is done, and it is achieved using the following two steps.

4.4.1 Manual Pre-Processing

In this part, some minor manual pre-processing is done to make the resultant CSV file into a more understandable format. For example, the cells with error, null, or any incredible value are replaced with "-1". In addition to that, the label column was added to the file, and records were labeled with their corresponding category.

4.4.2 Python Pre-Processing

After the manual preprocessing, we processed the dataset using python to prepare it for inputting the detection model. The preprocessing steps include:

1. One-Hot-Encoding Categorical Values: Categorical features usually have string values and are not numeric. As the created CSV file has many categorical feature values, such as header and text, it has to be transformed to a suitable format before learning models are able to process them. In order to achieve that, One-Hot-Encoding is done on the CSV file where each categorical feature is represented in the form of a vector, which the model can work with directly.

2. Feature Normalization: Normally, feature normalization is applied when feature values have different scales. This is done to allow all the features to follow a uniform distribution, making the values more comprehensible for the learning models. In this step, the Min-Max normalization is applied in the CSV file for our features.

4.4.3 Tools And Libraries

For modeling and preprocessing our CSV dataset, two python libraries such as pandas and Sklearn were used.

4.4.4 Coding Detail

To pre-process our dataset, categorical features are identified first and then encoded specifically, leaving the rest of the features with their original value as they were numeric. Once this is done, the encoded features are merged with the numeric ones. Figure 4.4 depicts the implementation of this process.

```
categorical_cols = ['images','text','header','stream','pages','pageno','xref',
'startxref','trailer','ObjStm','JS','Javascript','AA','OpenAction','Acroform',
'JBIG2Decode','RichMedia','launch','EmbeddedFile','XFA']
data_other_cols = X.drop(columns=categorical_cols)
print("one hot encoded")
one = OneHotEncoder(handle_unknown='ignore')
one.fit(X[categorical_cols].astype(str))
X = one.transform(X[categorical_cols].astype(str))
print(X.toarray())
# to dataframe
print("to dataframe")
X = pd.DataFrame(X.toarray())
```

Figure 4.4: Pre-Processing The Feature values

4.5 Detection Model Using Python

4.5.1 Tools And Libraries

In the detection model, in order to develop both single base learners and stacking models with different meta-learners, Sklearn libraries were used.

4.5.2 Coding Detail

The main code snippet for developing our stacking model is provided in Figure 4.5. In this, three different base learners and three meta models were used to evaluate the results, further elaborated in chapter 5.

```
#base models
estimators = [('svc',LinearSVC(dual=False)),
('rf', RandomForestClassifier(n_estimators=10, random_state=42)),
('mlp', MLPClassifier(hidden_layer_sizes=(20,5,20),alpha=0.005))]
strat_k_fold = KFold(n_splits=5, random_state=None, shuffle=True)

#meta model 1
clf = StackingClassifier(estimators=estimators,
final_estimator=LogisticRegression())
print(clf.score(X_test,y_test))
print("accuracy cross val score")
result = cross_val_score(clf, X, y, cv=strat_k_fold)
print(sum(result)/5)
print("F1 cross val score")
mean_score = cross_val_score(clf, X, y, scoring="f1_macro", cv=strat_k_fold).mean()
print(mean_score)

#meta model 2
clf = StackingClassifier(estimators=estimators,
final_estimator=KNeighborsClassifier(n_neighbors=3))
print(clf.score(X_test,y_test))
print("accuracy cross val score")
result = cross_val_score(clf, X, y, cv=strat_k_fold)
print(sum(result)/5)
print("F1 cross val score")
mean_score = cross_val_score(clf, X, y, scoring="f1_macro", cv=strat_k_fold).mean()
print(mean_score)

#meta model 3
clf = StackingClassifier(estimators=estimators,
final_estimator=DecisionTreeClassifier())
print(clf.score(X_test,y_test))
print("accuracy cross val score")
result = cross_val_score(clf, X, y, cv=strat_k_fold)
print(sum(result)/5)
print("F1 cross val score")
mean_score = cross_val_score(clf, X, y, scoring="f1_macro", cv=strat_k_fold).mean()
print(mean score)
```

Figure 4.5: Code Snippet For Stacking Implementation

4.6 Concluding Remarks

This chapter provided the implementation process for our detection framework and specified the code snippet to develop each module, along with the tools and libraries that we utilized. Moreover, we specifically presented the details of our feature extraction script (Achieved *Con5*), which extracts a set of 32 features from the input PDF files. The next chapter describes the motivation to generate a new dataset for PDF malware analysis and explains our creation procedure in detail.

Chapter 5

Generating New Dataset

As far as our knowledge goes, a good quality dataset is crucial for any data analysis task. This helps in improving the model's generalization as well as leading to more accurate analysis results. In this chapter, the available datasets for PDF malware detection are analyzed and eventually,a new dataset is created by ourselves for improving the old ones.

Majority of the previous researches proposing a PDF malware detection solution tested and trained their models on a well-known dataset of PDF Malware called "Contagio". This dataset is collected from the Contagio Malware Dump repository, provided by the External Data Source which consists of 11,173 malicious along with 9,000 benign files. A notable fact is that most of the previous studies using this dataset reported results with over 99% accuracy and such a high testing accuracy is usually not achieved with most of the datasets. This indicates that the proposed solutions are already so comprehensive that they are almost fully capable of detecting all malicious PDF files which contradicts as the malware PDF threat is still a remaining unhandled issue. This was the motivation for us to analyze Contagio dataset further to lead to some clues behind the unrealistic high testing accuracy, with the aim of generating a new baseline dataset that overcomes the identified issues

of Contagio dataset.

5.1 Evaluating The Available Dataset

This section presents our analysis and evaluation results on the Contagio dataset in terms of some primary quality criteria that should be met when creating a dataset. The following are two significant issues that we found in the Contagio Dataset. (Achieved *Con3*)

All the presented issues are based on analyzing the CSV file that we generated by extracting 32 features from each of the PDFs in the Contagio repository.

1. Duplicate Entries: The 32 features are extracted from each PDF file in the repository and based on those features, a high number of duplicate entries are found in the resultant CSV dataset.Moreover, it was identified that there exists 8,977 duplicate entries in this dataset, which accounts for 44% of the entire dataset. This indicates a very high similarity between PDF file sample structures used in the Contagio repository.

2. Lack Of Coverage: A good dataset should cover multiple varieties of the samples of interest. In the case of PDF files, it must include certain samples that contain certain features which are diverse among each of the classes of the dataset. As an example, coverage of evasive samples, which can be defined as benign samples that are very similar to malicious ones or vice versa, is one of the main concerns. In order to evaluate the variety of the data points in the Contagio dataset,a simple metric referred to as coefficient of variation is applied. The Coefficient of Variation (CV) is defined as the ratio of standard deviation to the mean of a set of values, and it is used to measure the relative variability of a set of values. According to wikipedia a CV higher than 1 indicates a relatively high variance, whereas a value less than 1 denotes a

low variance among data points. Based on this, we applied the CV formula to each of the columns in the dataset which represent each feature value and calculated the average of all the results to be able to represent the entire dataset variability with a single number. As a result we obtained the value of 0.83 for malware samples and the value of 0.79 for the benign ones. Since both values are lower than 1, they are considered as low variances which indicates that there is a low variation among both malicious and benign PDF file classes and the result coordinates with what we expected before running our analysis.

As discussed in chapter 2 and 3, one of the most typical methods to deliver a malware through PDF files is executing a malicious script upon opening the PDF. This means that in typical PDF attacks two powerful PDF features ("Javascript" and "OpenAction") are commonly exploited. In order to further analyze the lack of coverage in Contagio , we considered this as well. In the Contagio dataset, we observed that more than 0.74 of the malicious data points have a value of more than 0 for the mentioned two features, whereas only 0.0007 of the benign files have a value of more than 0 for both. This makes the samples almost easily detectable solely on the basis of two features, and it enables a simple linear classifier to produce a minimum accuracy of around (80%) purely based on the two specified features. This does not well represent the real-world PDF malware exploitation as attacks are getting more and more sophisticated and attackers exploit various other features instead of these two to maximize their chances of evasion and hinder being easily detected. Therefore, this is another clear indication of lack of comprehensiveness and propoer malware distribution in Contagio dataset.

5.2 Creating An Evasive PDF Dataset

By assessing the quality of data in the Contagio dataset, some primary issues were identified which makes the dataset less reliable to be used as an evaluation basis. Therefore, we decided to generate a new dataset to minimize the issues specified for the Contagio dataset and obtain more realistic results. This section presents more details on how we created a second dataset to use as another baseline for our evaluation.

Our main methodology for building the new dataset was to combine the available datasets in a way that the samples meet the criteria that we are looking for. For achieving that, 11,173 malicious files from Contagio and 20,000 malicious files from VirusTotal along with 9,109 benign files from Contagio were collected. Then, the features are extracted from all these PDFs using the developed source code and stored the results in different csv files. The final goal in the work is to wisely combine the two datasets' records into one final file, which results in a more representative dataset of the PDF distribution and for that, before combining the csv files, we first removed all the duplicate records in each of them.

Table 5.1 depicts what we are left with after this procedure.

As we can see, a huge proportion of the records are removed after deduplication, and a much smaller dataset is left as our resource :

Table 5.1: Available dataset resources for generating our new dataset

Dataset	Without Duplicate Removal	After Duplicate Removal
Contagio Malicious Samples	11,173	3,223
VirusTotal Malicious Samples	20,096	4,868
Contagio Benign Samples	9,000	8,268

To tackle the coverage issue of the Contagio dataset, we decided to increase the ratio of evasive samples in both categories of PDF and to achieve this, we decided to employ K-means, an unsupervised machine learning used to cluster the data into k groups by their similarity.

First, we clustered each of the available CSV datasets into two groups by their similarity in the following steps :

1. On each malware CSV file, we clustered the data samples by the similarity between all of their 31 features. The samples falling into the wrong cluster with the malicious label are taken as an evasive set of malicious records. This is done based on the intuition that the features of these samples were not so similar with the rest of them to be clustered in the right group.

2. For obtaining the second set of evasive samples, the data samples are clustered on each malicious dataset once again. However, this time, we did not consider the whole feature set by choosing to limit the features to only 10 General ones and separated the samples falling into the wrong cluster with the malicious label.

3. Steps 1 and 2 are repeated for discovering evasive samples among benign records as well.

4. Finally, we added a set of 109 benign PDF files with special effects and featured not commonly found in typical benign PDF samples. These are also taken from the Contagio repository, and we believe they are a good representation of evasive benign PDF samples because of their specific properties.

Throughout the above procedures, all the duplicate values were removed and we ended up with our new dataset with a reasonable size, consisting of 10,025 samples. The source code for building our dataset is provided in Appendix A.

Table 5.2 further illustrates our dataset distribution in more details and we refer to this newly created dataset as our dataset in the following sections. (Achieved *Con4*)

Table 5.2: Our dataset specifications

Total Records	Malicious Records	Benign Records
10,025	5,557	4,468

5.3 Evaluating The New Dataset

To prove the effectiveness of our approach for building a better dataset, we conducted certain experiments to evaluate and compare it with the old one. The objective of conducting these experiments is to verify that our new dataset has improved the quality of Contagio that aims to address the two main issues with Contaigo dataset. Below we provided the justification on how each of these issues is handled in the new dataset.

1. Duplicate entries issue: The first issue that we aimed to address to increase our dataset quality was to reduce the number of duplicate records. As discussed in our dataset creation procedure, all the duplicate records of both source datasets are removed before extracting our interests records. Hence our dataset consists of 10,022 samples with no duplicate records.

2. Lack of coverage issue: To evaluate how successfully we could tackle the coverage issue and how diverse the new dataset is compared to the old one, we decided to utilize two metrics, Standard Deviation and Variance. These metrics are used to measure how spread out a set of values are from each other and are computed as described in Equation (1).

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N}}$$
$$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N} \quad (1)$$

They can be applied to represent the variability of the data points in a dataset. In this fashion, we decided to calculate these metrics once based on the whole set of our features and another time based on the general features only. Since each of our data points consist of multiple columns(features), we computed the standard deviation and variance for the values of each column separately and computed their average as the final results.

(a) Variance And Standard Deviation Based On All Features: This evaluation comparison is to measure the diversity of our data samples based on the whole feature set. We provided the result comparison between benign samples of each dataset and the same results for malicious samples in Table 5.3. The outputs clearly show that both variance and standard deviation are considerably higher in our dataset than Contagio. This can be seen both among the benign samples and malicious ones. It is also observed that, in general, diversity among benign samples is less than malicious ones in both datasets.

(b) Variance And Standard Deviation Based On General Features: The purpose of making this evaluation comparison is to show how diverse the PDF files are, solely based on their general features such as their size and pages. We seek to achieve a higher number, as it indicates that the same group files are harder to detect by general features only. We provided the result comparison between benign samples of each dataset and the same results for malicious samples in Table 5.3. As demonstrated, our dataset's variety is remarkably greater than Contagio, which is a good indication that it will be much more difficult to classify our samples based

on general features only. Another important observation is the fact that, in general, diversity tends to be significantly higher when it is based on general features compared to the whole features. This shows that our extracted features are not redundant, and in fact, adding them is making the dataset more homogeneous.

(c) Ratio of PDF files having Javascript and OpenAction features: Another shortcomings of Contagio in terms of lack of sufficient coverage was that a large portion of all malicious files (74%) contained one or more "Javascript" and 'OpenAction' features, whereas only a tiny fraction of benign files (0.007) had both two features together. In our new dataset, this ratio is more realistic and balanced at around 49% , which shows that attack varieties are less limited to exploiting two features only.

Table 5.3: Variance and Standard Deviation Comparison Results

Group	Dataset	Variance	Standard Deviation
Benign With All Features	New Contagio	9965.58 591.11	28.27 11.39
Malicious With All Features	New Contagio	343595.33 7098.43	190.77 23.79
Benign With General Features	New Contagio	28610.61 1648.5	76.29 27.3
Malicious With General Features	New Contagio	286987826.99 20405.42	545.23 67.74

5.4 Concluding Remarks

In this chapter, we highlighted the main flaws observed in the Contagio dataset based on our theoretical and experimental analysis (Achieved *Con3*). Furthermore, we addressed these flaws by creating a new baseline dataset named "Evasive PDF

Dataset" that produced more realistic and reliable results (Achieved *Con4*). In addition, the improvement of our dataset is demonstrated through a set of evaluation criteria. In the next chapter, we finalize our proposed model structure through experiment and we present and analyze the comparison results.

Chapter 6

Experiments and Results

In this chapter we present our experiments to finalize our proposed model, tune its parameters and then evaluate our model on two different datasets: Contagio dataset and the newly created Evasive PDF dataset.

6.1 Data Repository

In this research, we mainly used the Contagio dataset to finalize our proposed model's structure and compare our results with other research works. Moreover, results are tested on the second dataset, labeled as "Evasive PDF Dataset", to further confirm our model's effectiveness. In the following sub sections, we review each specification of the dataset.

6.1.1 Contagio Dataset Distribution

The Contagio dataset consists of 9000 benign files and 11,173 malicious files as illustrated in the Table 6.1.

As these PDF files are not ready to be fed into the learning system as they are, we apply the feature extraction code that included in the implementation chapter. Once the code is applied, a CSV file is created, consisting of 20,109 entries with 32

Table 6.1: Contagio Dataset distribution

Total Samples	Benign Samples	Malicious Samples
20,173	9,000	11,173

features. However, we could not extract features from 64 of the malicious files and we ended up with 20,109 records with 33 columns where 32 are extracted features and one column as the label.

Table 6.2: Our generated csv data file from Contagio specifications

Total Records	Benign Records	Malicious Records	Features
20,109	9,000	11,109	32

6.1.2 Evasive PDF Dataset Distribution

The is the dataset which we previously built in chapter 5 that consists of 10025 evasive PDF files as illustrated in the Table 6.3.

Table 6.3: dataset distribution

Total Samples	Benign Samples	Malicious Samples
10,025	5,557	4,468

6.2 Finalizing The Proposed Model

In this section, we aim to finalize the structure of our proposed model based on the performance results, through several tests. As we already shortlisted four different baseline classifiers and three different meta-learners to form our detection model, we first tune the hyper-parameters of each of the base-learners, and test all four base-learners and analyze their performance. Once this is done, we evaluate different

stacking combinations by applying each of the selected meta-learners in chapter3, and present each result.

6.2.1 Dataset Split

As specified in chapter 3, 5-Fold cross-validation is selected to carry out all of our evaluations, which is a 80/20 train-test split ratio for each validation.

6.2.2 Tuning Base-Learner Parameters

First, before defining the type of meta-learners and selecting the right combination of base-learners,the base learners' parameters are tuned and based on the experimental results, we finalized those values for each of the base learners. Below we present our final selected values for each hyper-parameter without comparing the unnecessary details of each single experiment.

1. Random Forest Estimators: Number of trees in a random forest is the most important hyper-parameter that has to be determined. In chapter 3, we theoretically justified that a number between 10-50 would be a suitable number of estimators, and through experimentation, we were able to achieve the highest accuracy and F1 score, using 20 estimators.

2. MLP Nodes and Layers: The number of layers and nodes at each MLP layer are crucial hyper-parameters. Hence, we theoretically proposed two potentially suitable values as the number of hidden nodes according to thumb rule. Results showed that the best performance is achieved when three hidden layers are used with 20, 10, and 20 nodes at each layer.

6.2.3 Finalizing Learning Detection Model

After determining the base-learner parameters, the second round of the experiments is started to determine the best stacking combination and the best meta-learner to build our detection model.

Chapter 3 finalized main parameters such as the number of base-learners, the number of meta-learners, and the number of stacking levels based on theory. However, we left a small portion for experimentation: choosing the best subset of the shortlisted base-learners and the best meta-learner out of the three proposed meta-learners in chapter 3.

For this, each individual base-learner is tested on the baseline dataset and reported the results as shown in the Table 6.4 and Figure 6.1. In order to generate reliable testing scores, all the results are provided using 5-fold cross-validation and finally, the results are analyzed in terms of 4 common metrics: accuracy, F1 score, recall and precision.

Table 6.4: Base-Learner Results

Algorithm	Accuracy	Precision	Recall	F1 score
Random Forest	99.87	99.84	99.86	99.86
MLP	99.82	99.83	99.83	99.82
Linear SVM	99.83	99.83	99.83	99.85
Adaboost	99.77	99.74	99.73	99.75

It is observed that all four classifiers have a quite high accuracy and F1 score which supports our theory about the fitness of these classifiers in chapter 3. We can also observe that the results are quite similar, and their differences are very marginal. We could also identify that the highest scores belong to Random Forest whereas Adaboost has the lowest performance. Surprisingly MLP and Linear SVM performed at the same level, with MLP taking much longer to train.

To select the most suitable meta-model, we choose the one that produces the best

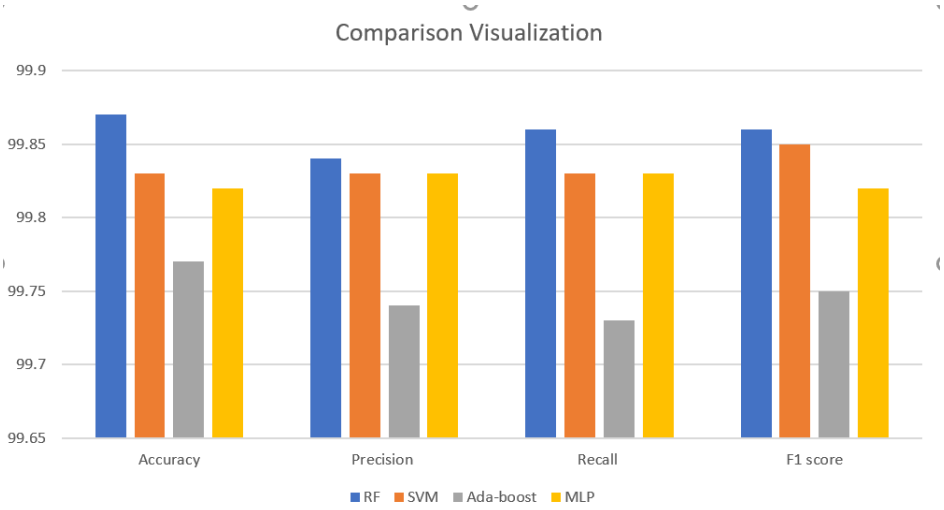


Figure 6.1: Base learner Results Visualization

results for our proposed stacking model and tried stacking multiple combinations of the base-learners. Finally, we obtained the best results by combining 'MLP', 'Linear SVM' and 'RF', and provided the results by applying different meta-models as shown in the Table 6.5 and visualized in the Figure 6.5.

It is also identified that using Logistic Regression as the meta-learner provides the best results, which is higher than the individual scores from each candidate base-learner. Hence, from now, this is the result we refer to compare our performance with relevant studies and our final proposed model architecture is illustrated in Figure 6.2 .

Table 6.5: Proposed model results with different meta-learners

Metamodel	Accuracy	Precision	Recall	F1 score
Logistic Regression	99.89	99.84	99.88	99.86
K Nearest Neighbors	99.86	99.81	99.82	99.81
Decision Tree	99.85	99.75	99.70	99.78

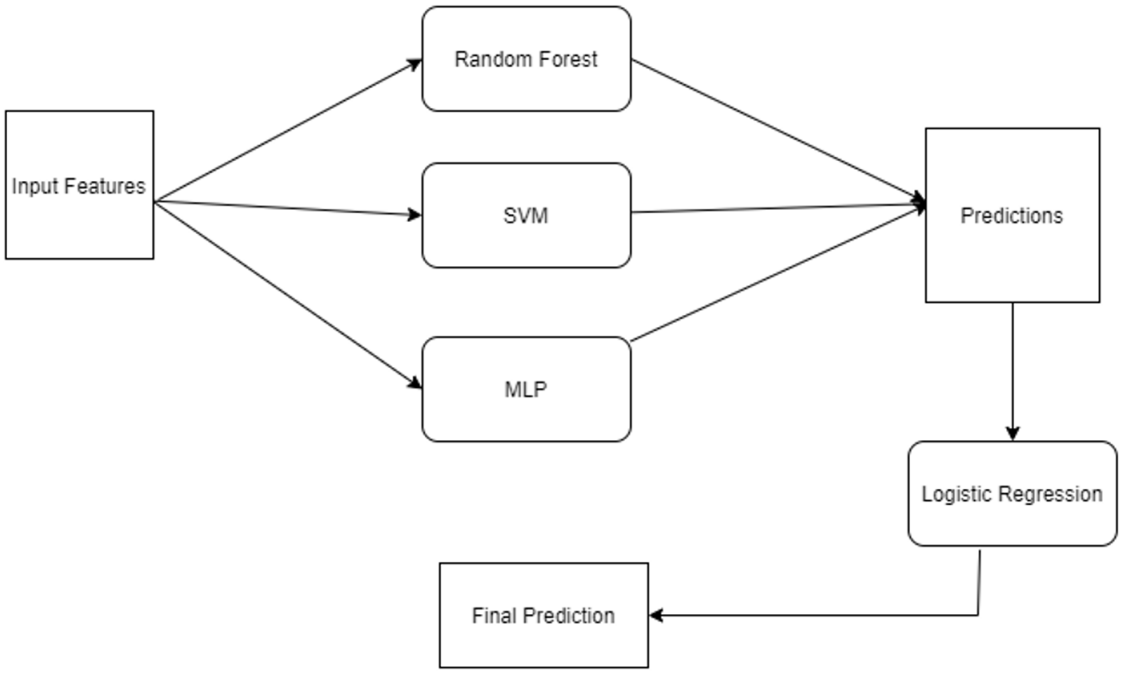


Figure 6.2: Final proposed stacking architecture

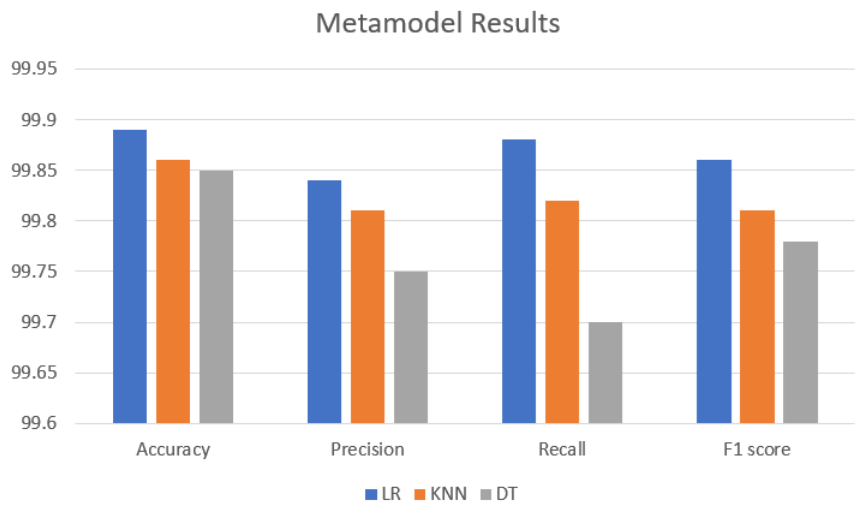


Figure 6.3: Metamodel Results Visualization

6.3 Evaluating The Proposed Model

This section evaluates our finalized model performance by first testing it on the Contagio dataset to compare the results with previous relevant works. Moreover, we

re-evaluate it on the Evasive PDF Malware Dataset, which is created in this research.

6.3.1 Comparing Proposed Model With Relevant Studies

It is generally a challenging task to compare our proposed method with different relevant works as some of the studies did not provide all the measurements, and majority did not specify their validation and evaluation details, such as the training and testing split ratios. Most importantly, it was difficult to access the exact dataset they trained and tested their model with. However, we tried to shortlist some papers that purely used Contagio dataset and compare their results in the Table 6.6. Eventhough the specified research works were solely relied on Contagio dataset to evaluate their research, the evaluation method was not specified. Based on the results, it was identified that our approach achieves a higher testing accuracy than the other two researches. (Achieved *Con1*)

Table 6.6: Comparison of our work with relevant studies

Research	Accuracy	Precision	Recall	F1 score
[22]	99.68	not provided	not provided	not provided
[24]	99.83	not provided	not provided	not provided
Proposed Model	99.89	99.84	99.89	99.86

6.3.2 Evaluating The Proposed Model Using New Dataset

Our dataset proved to be more reliable than Contagio dataset based on a set of criteria. We decided to evaluate our model once again on our own dataset to validate its effectiveness further. We can observe the results in 6.8, which are presented using 5-fold cross-validation. Comparing the individual scores of each classifier presented in Table 6.7, we can see that our proposed model outperforms each of the individual scores, which further verifies the robustness of our proposed model.

Table 6.7: base-learner results on the new dataset

Algorithm	Accuracy	Precision	recall	F1 score
Random Forest	98.44	98.65	98.68	98.66
MLP	98.33	98.55	98.53	98.52
Linear SVM	98.21	98.23	98.23	98.28
Adaboost	97.32	97.75	99.73	97.75

Table 6.8: Proposed model results on our dataset

Accuracy	Precision	Recall	F1 score
98.69	98.88	98.87	98.77

6.4 Other Findings

In the previous sections, we analyzed our proposed solution in terms of main performance metrics. However, criteria such as implementation cost, computational complexity, and ease of deployment are also important when developing a comprehensive solution. Here, we present some further analysis of our proposed solution with the aid of our experimental results.

One of the strengths of our solution is that it relies on a small set of 32 features for achieving the specified results and all these features are among structural and general PDF properties. This has a number of advantages such as contribution to the efficacy and speed of feature extraction as the feature extraction requires minimal content parsing and decoding, and they can be extracted in a more reliable fashion than content features as discussed in the section 2.2.3.1. This makes our solution less complex and easier to deploy.

Although some might argue that using our proposed set of features makes our detection solution easier to evade by attackers, our robust stacking-based detector counters this issue as it aggregates the predictions of three different classifiers with a totally different outlook on the problem thus minimizing the evasion opportunities

for attackers.

Additionally, we measured the time consumption of our detection model and incurs a computational training overhead of 25 seconds, which is a relatively large number. This is due to the fact that the stacking requires all the base classifiers to be cross-validated on the entire dataset set first. However, the proposed model only incurs a negligible amount of 0.6 seconds for prediction. Lastly, it is noteworthy that our proposed solution has a relatively low cost of implementation and minimal resource requirements, as it does not involve any simulation or code execution.

6.5 Concluding Remarks

This chapter provided the experimentation details for determining our final proposed model and then compared the selected model's performance with other relevant works. Furthermore, we explored stacking as a method for malicious PDF prediction and observed that by using the suitable architecture, we are able to improve the results of single classifiers. Based on the results, we could identify that the best results are achieved by combining SVM, Random Forest, and MLP as base learners and Logistic Regression as the meta-learner. This framework also allowed us to successfully improve the accuracy and F1 score results of previous works (Achieved *Con1*). Finally, we re-evaluated our proposed model on our new dataset and obtained improved performance results, further validating our model as a potentially effective solution for this problem. In the next chapter, we present our overall conclusion and discuss our future works.

Chapter 7

Conclusion And Future Work

7.1 Conclusion

Malicious PDF files are a severe and prevalent Cyber risk in today's world where PDF files are widely used. Analyzing the PDF complex structure and powerful features, we can conclude that malware authors can deliver malware in multiple ways. Furthermore, the lack of user awareness coupled with the ineffectiveness of common anti-viruses has increased this risk even more. Based on the studies, we presented the vast attack vectors in our taxonomy at the end of chapter 2.

Several solutions have been proposed by far for PDF malware detection. The majority rely on static-based approaches and Machine Learning technologies as their main detection schema, with their strengths and weaknesses. In this research, we proposed a novel stacking-based framework for detecting malware PDF files. We demonstrated our solution's effectiveness through several experimentations by extracting a set of 31 representative features through our developed feature extraction script and stacking three different algorithms.

Furthermore, we generated a new dataset according to specific data quality criteria that represented the real-world distribution of benign and malicious PDF files,

leading to more reliable results.

7.2 Future Work

There are various directions to extend our research further, including:

1. Exploring other levels of stacking: In this study, we improved results by using two levels of stacking. As we did not explore deeper levels to maintain a lightweight solution and avoid higher time complexity, a new hardware advancement with more profound stacking levels can be considered a potential solution.

2. Improving our newly generated dataset: Our newly generated dataset can be further improved by gathering more malicious and benign samples from diverse resources. Moreover, we can increase the dataset size by manually creating new PDF files and inserting features such as JavaScript or embedding media within them.

3. Extracting the rest of our proposed features: In this work, we shortlisted 37 features and excluded five of them due to the cost ineffectiveness of extraction. However, as a part of our future work, we plan to extract the rest of the features to compare the results.

Bibliography

[1] *Adobe help center*, https://helpx.adobe.com/support.html, (Accessed on 10/27/2021).

[2] *Akamai blog — over 25% of malicious javascript is being obfuscated*, https://www.akamai.com/blog/security/over-25-percent-of-malicious-javascript-is-being-obfuscated, (Accessed on 10/21/2021).

[3] *Automated malware analysis tool — falcon sandbox — crowdstrike*, https://www.crowdstrike.com/endpoint-security-products/falcon-sandbox-malware-analysis/, (Accessed on 10/22/2021).

[4] *Cuckoo sandbox - automated malware analysis*, https://cuckoosandbox.org/, (Accessed on 10/22/2021).

[5] *Cuckoo sandbox book*, https://buildmedia.readthedocs.org/media/pdf/cuckoo/, (Accessed on 10/22/2021).

[6] *Excel malware targets business & home users - the excel experts*, https://www.theexcelexperts.com/how-microsoft-excel-is-being-used-to-spread-malware/, (Accessed on 10/22/2021).

[7] *Free automated malware analysis service - powered by falcon sandbox*, https://www.hybrid-analysis.com/sample/, (Accessed on 10/22/2021).

[8] *Leveraging falcon sandbox to detect and analyze malicious pdfs containing zero-day exploits*, https://www.crowdstrike.com/blog/leveraging-falcon-sandbox-to-detect-and-analyze-malicious-pdfs-containing-zero-day-exploits/, (Accessed on 10/22/2021).

[9] *Phishing attacks are what percentage of cyber attacks?*, https://www.safeguardcyber.com/blog/phishing-attacks-are-what-percentage-of-cyber-attacks, (Accessed on 10/06/2021).

[10] *Phishing trends with pdf files in 2020: 5 approaches attackers use*, https://unit42.paloaltonetworks.com/phishing-trends-with-pdf-files/, (Accessed on 10/05/2021).

[11] *What to do if you click on a phishing link? - the defence works*, https://thedefenceworks.com/blog/what-to-do-if-you-click-on-a-phishing-link/, (Accessed on 10/06/2021).

[12] *Will new authentication technology spell the end of spam? - techrepublic*, https://www.techrepublic.com/article/will-new-authentication-technology-spell-the-end-of-spam/, (Accessed on 10/21/2021).

[13] Amir Afianian, Salman Niksefat, Babak Sadeghiyan, and David Baptiste, *Malware dynamic analysis evasion techniques: A survey*, ACM Computing Surveys (CSUR) **52** (2019), no. 6, pp. 1–28.

[14] Barracuda, *Protecting your business from cyberthreats in 2020, how to fight phishing, ransomware and other top attack*, https://assets.barracuda.com/assets/docs/dms/eBook$_t hreat - spotlight - 2020.pdf$, $(Accessedon 09/10/2021)$.

[15] Ruth Bearden and Dan Chai-Tien Lo, *Automated microsoft office macro malware detection using machine learning*, 2017 IEEE International Conference on Big Data (Big Data), IEEE, 2017, pp. pp. 4448–4452.

[16] Alexandre Blonce, Eric Filiol, and Laurent Frayssignes, *Portable document format (pdf) security analysis and malware threats*, Presentations of Europe Black-Hat 2008 Conference, 2008.

[17] Marcus Botacin, Felipe Duarte Domingues, Fabrício Ceschin, Raphael Machnicki, Marco Antonio Zanata Alves, Paulo Lício de Geus, and André Grégio, *Antiviruses under the microscope: A hands-on perspective*, Computers & Security (2021), 102500.

[18] Hongming Che, Qinyun Liu, Lin Zou, Hongji Yang, Dongdai Zhou, and Feng Yu, *A content-based phishing email detection method*, 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), IEEE, 2017, pp. pp. 415–422.

[19] Aviad Cohen, Nir Nissim, Lior Rokach, and Yuval Elovici, *Sfem: Structural feature extraction methodology for the detection of malicious office documents using machine learning methods*, Expert Systems with Applications **63** (2016), pp. 324–343.

[20] Igino Corona, Davide Maiorca, Davide Ariu, and Giorgio Giacinto, *Lux0r: Detection of malicious pdf-embedded javascript code through discriminant analysis of api references*, Proceedings of the 2014 workshop on artificial intelligent and security workshop, 2014, pp. pp. 47–57.

[21] Jesse S Cross and M Arthur Munson, *Deep pdf parsing to extract features for detecting embedded malware*, Sandia National Labs, Albuquerque, New Mexico, Unlimited Release SAND2011-7982 (2011).

[22] Bonan Cuan, Aliénor Damien, Claire Delaplace, and Mathieu Valois, *Malware detection in pdf files using machine learning*, (2018).

[23] SL Shiva Darshan, MA Ajay Kumara, and CD Jaidhar, *Windows malware detection based on cuckoo sandbox generated report using machine learning algorithm*, 2016 11th International Conference on Industrial and Information Systems (ICIIS), IEEE, 2016, pp. pp. 534–539.

[24] Raphael Fettaya and Yishay Mansour, *Detecting malicious pdf using cnn*, arXiv preprint arXiv:2007.12729 (2020).

[25] Nicolas Fleury, Theo Dubrunquez, and Ihsen Alouani, *Malware: An overview on threats, detection and evasion attacks*, arXiv preprint arXiv:2107.12873 (2021).

[26] Jacob Gajek, *Macro malware: dissecting a malicious word document*, Network Security **2017** (2017), no. 5, pp. 8–13.

[27] Simon Huneault-Leblanc and Chamseddine Talhi, *P-code based classification to detect malicious vba macro*, 2020 International Symposium on Networks, Computers and Communications (ISNCC), IEEE, 2020, pp. pp. 1–6.

[28] Sainadh Jamalpur, Yamini Sai Navya, Perla Raja, Gampala Tagore, and G Rama Koteswara Rao, *Dynamic malware analysis using cuckoo sandbox*, 2018 Second international conference on inventive communication and computational technologies (ICICCT), IEEE, 2018, pp. pp. 1056–1060.

[29] Young-Seob Jeong, Jiyoung Woo, and Ah Reum Kang, *Malware detection on byte streams of pdf files using convolutional neural networks*, Security and Communication Networks **2019** (2019).

[30] Sangwoo Kim, Seokmyung Hong, Jaesang Oh, and Heejo Lee, *Obfuscated vba macro detection using machine learning*, 2018 48th annual ieee/ifip international

conference on dependable systems and networks (dsn), IEEE, 2018, pp. pp. 490–501.

[31] Renuka Kumar and Anand Raj Essar Vaishakh, *Detection of obfuscation in java malware*, Procedia Computer Science **78** (2016), 521–529.

[32] Yuanzhang Li, Yaxiao Wang, Ye Wang, Lishan Ke, and Yu-an Tan, *A feature-vector generative adversarial network for evading pdf malware classifiers*, Information Sciences **523** (2020), pp. 38–48.

[33] Chih-Hung Lin, Chin-Wei Tien, Chih-Wei Chen, Chia-Wei Tien, and Hsing-Kuo Pao, *Efficient spear-phishing threat detection using hypervisor monitor*, 2015 International Carnahan Conference on Security Technology (ICCST), IEEE, 2015, pp. 299–303.

[34] Daiping Liu, Haining Wang, and Angelos Stavrou, *Detecting malicious javascript in pdf through document instrumentation*, 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, IEEE, 2014, pp. 100–111.

[35] Zhongjin Liu, Le Zhang, Qiuying Ni, Juntai Chen, Ru Wang, Ye Li, and Yueying He, *An integrated architecture for iot malware analysis and detection*, International Conference on Internet of Things as a Service, Springer, 2018, pp. pp. 127–137.

[36] Davide Maiorca, Davide Ariu, Igino Corona, and Giorgio Giacinto, *A structural and content-based approach for a precise and robust detection of malicious pdf files*, 2015 international conference on information systems security and privacy (icissp), IEEE, 2015, pp. 27–36.

[37] Davide Maiorca, Battista Biggio, and Giorgio Giacinto, *Towards adversarial malware detection: Lessons learned from pdf-based attacks*, ACM Computing Surveys (CSUR) **52** (2019), no. 4, pp. 1–36.

[38] Timothy McIntosh, Paul Watters, ASM Kayes, Alex Ng, and Yi-Ping Phoebe Chen, *Enforcing situation-aware access control to build malware-resilient file systems*, Future Generation Computer Systems **115** (2021), pp. 568–582.

[39] Mamoru Mimura, *An improved method of detecting macro malware on an imbalanced dataset*, IEEE Access **8** (2020), pp. 204709–204717.

[40] Hiroya Miura, Mamoru Mimura, and Hidema Tanaka, *Discovering new malware families using a linguistic-based macros detection method*, 2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW), IEEE, 2018, pp. pp. 431–437.

[41] Tianshi Mu, Huajun Chen, Jinran Du, and Aidong Xu, *An android malware detection method using deep learning based on api calls*, 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), IEEE, 2019, pp. pp. 2001–2004.

[42] Hiran V Nath and Babu M Mehtre, *Static malware analysis using machine learning methods*, International Conference on Security in Computer Networks and Distributed Systems, Springer, 2014, pp. 440–450.

[43] Ani Nenkova, Jieun Chae, Annie Louis, and Emily Pitler, *Structural features for predicting the linguistic quality of text*, Empirical methods in natural language generation, Springer, 2009, pp. pp. 222–241.

[44] Uğur Ozker and Ozgur Koray Sahingoz, *Content based phishing detection with machine learning*, 2020 International Conference on Electrical Engineering (ICEE), IEEE, 2020, pp. pp. 1–6.

[45] Priyansh Singh, Shashikala Tapaswi, and Sanchit Gupta, *Malware detection in pdf and office documents: A survey*, Information Security Journal: A Global Perspective **29** (2020), no. 3, pp. 134–153.

[46] Didier Stevens, *Pdf, let me count the ways*, https://blog.didierstevens.com/2008/04/29/, (Accessed on 10/21/2021).

[47] Jose Torres and Sergio De Los Santos, *Malicious pdf documents detection using machine learning techniques*, Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018), 2018, pp. 337–344.

[48] Mihai Vasilescu, Laura Gheorghe, and Nicolae Tapus, *Practical malware analysis based on sandboxing*, 2014 RoEduNet Conference 13th Edition: Networking in Education and Research Joint Event RENAM 8th Conference, IEEE, 2014, pp. pp. 1–6.

[49] Christian Wressnegger, Frank Boldewin, and Konrad Rieck, *Deobfuscating embedded malware using probable-plaintext attacks*, International Workshop on Recent Advances in Intrusion Detection, Springer, 2013, pp. pp. 164–183.

[50] Jason Zhang, *Mlpdf: an effective machine learning based approach for pdf malware detection*, arXiv preprint arXiv:1808.06991 (2018).

[51] Mi ZhiWei, Manmeet Mahinderjit Singh, and Zarul Fitri Zaaba, *Email spam detection: a method of metaclassifiers stacking*, The 6th international conference on computing and informatics, 2017, pp. pp. 750–757.

Appendix A

This is Appendix A

from sklearn.cluster import KMeans

from matplotlib import pyplot

import matplotlib.pyplot as plt

import pandas as pd

define dataset

from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, LabelEncoder

data1 = pd.read$_csv(r'Contiagopdfdatasetcsvpdfdataset$

$output.csv', dtype =' unicode')$

data1 = data1.astype(str)

data1 = data1.dropna()

data1 = data1.drop$_duplicates()$

#pre-processing

oe = LabelEncoder()

#data1 = data1.astype(str)

data1 = data1.reset$_index()$

data = data1.apply(LabelEncoder().fit$_transform)$

X = data[['pdfsize','stream','obj','ObjStm','EmbeddedFile','Javascript','OpenAction','launch','AA

'header','pages','metadata size','title characters','images','text','isEncrypted']]

defining the model

model = KMeans(n$_c lusters = 2).fit(X) centeroids = model.cluster$_c enters_p rint(centeroids)$

plt.scatter(centeroids[:,0],centeroids[:,1],c='red', s=50)

plt.show()

#writing down the results on a csv file

cluster$_r esult = pd.DataFrame()$

cluster$_r esult['index'] = X.index.values$

cluster$_r esult['cluster'] = model.labels$

cluster$_r esult['label'] = data1.label$

finding evasive samples

result = cluster$_r esult[(cluster_r esult['cluster'] == 0)(cluster_r esult['label'] ==' 0')]$

print("cluster 0 label 0")

data1.iloc[result.index.values].to$_c sv(r'evasion2mal.csv')$

result = cluster$_r esult.loc[(cluster_r esult['cluster'] == 1)(cluster_r esult['label'] ==' 0')]$

print("cluster 1 and label 0")

print(data1.iloc[result.index.values])

data1.iloc[result.index.values].to$_c sv(r'evasion2mal.csv')$

result = cluster$_r esult.loc[(cluster_r esult['cluster'] == 0)(cluster_r esult['label'] ==' 1')]$

print("cluster 0 and label 1")

data1.iloc[result.index.values].to$_c sv(r'evasion1ben.csv')$

result = cluster$_r esult.loc[(cluster_r esult['cluster'] == 1)(cluster_r esult['label'] ==' 1')]$

print("cluster 1 label 1")

data1.iloc[result.index.values].to$_c sv(r'evasion2ben.csv')$

```
print(result)
```

Vita

Candidate's full name: Maryam Issakhani
University attended:

- M.Sc. degree in Computer Science in Cybersecurity field, under the supervision of professor Arash Habibi Lashkari, University of New Brunswick (UNB), Winter 2021-Fall 2021
- B.Sc. degree in Computer Engineering, Alzahra University, Iran, Fall 2015-Summer 2019

Publications: A conference paper has been submitted to ICISSP2021 entitled "A Novel Evasive PDF Malware Detection Model Based on Stacking Learning"