DOI: https://doi.org/10.67897/njma.2022.4rcm0i4v



Nigerian Journal of Mathematics and Applications
Volume **32**, (2022), 117 – 135. *Printed by Unilorin press*
©Nig. J. Math. Appl. *http : //www.njmaman.com*

Malicious PDF Detection Using Support Vector Machine

G. B. Balogun*, P. F. Adinoyi, J. B. Awotunde, M. Abdulraheem and I. D. Oladipo

Abstract

The aim of this study is to develop a system that can detect and classify the Portable Document Format (PDF) as malware or benign document based on its features, using the Support Vector Machine (SVM) as an underlying model. Based on feature extraction and the Support Vector Machine technique, a method is proposed for efficiently detecting PDFs with harmful payloads. It was proved in this study that by extracting a wide range of feature sets, a robust PDF malware classifier can be produced. By employing data sets with a total of 10,000 harmful and 10,000 benign documents, the classification rate can reach up to 99 percent while retaining low false positive rates of 0.2 percent or less for various classification parameters.

1. Introduction

Malware-bearing programs and documents continue to make headlines. Criminal organizations, companies, and governments frequently employ them to steal money, exploit users, eavesdrop, and engage in other unlawful actions. Malicious codes are frequently inserted or bundled within documents or programs that appear legal; as a result, these documents are Trojan Documents because they contain a dangerous payload in a seemingly appealing document that serves as a

Received: 01/04/2022, Accepted: 18/06/2022, Revised: 30/06/2022. * Corresponding author.
2015 *Mathematics Subject Classification.* 94A13 & 94B60.
Keywords and phrases. Malware, Benign, JavaScript, Code, Support Vector Machine
Department of Computer Science, University of Ilorin, Ilorin, Nigeria
E-mails of the corresponding author: balogun.gb@unilorin.edu.ng
ORCID of the corresponding author: 0000-0003-3478-4854

malware distribution vector [16]. Many recent researches have shown that harmful papers are routinely employed in highly socially engineered phishing assaults carried out by groups of very clever, persistent, and targeted attackers with an espionage purpose [2].

The Portable Document Format (PDF) file format was created in 1993 and has been maintained by Adobe Software for the past years. Unfortunately, PDF has become a popular conduit for user exploitation ranging from large-scale phishing operations to targeted attacks due to its multiple capabilities and widespread use. Despite recent breakthroughs in machine learning for malware detection, antivirus firms still rely primarily on handwritten signatures to detect malicious PDFs. Not only does this require a considerable number of human resources, but it is also rarely useful in detecting unknown variants or zero-day assaults.

Because of its numerous features and extensive use, the Portable Document Format (PDF) has become a popular conduit for user exploitation ranging from large-scale phishing operations to targeted attacks. According to Charles and Angeles (2012), the use of Portable Document Formats (PDFs) as a vehicle for user exploitation, as well as the exploitation of vulnerabilities in client applications such as document viewers, has grown in popularity. Furthermore, social engineering makes convincing consumers to view a malware-infected document easier. Exploits operate by exploiting a weakness in a PDF reader application's API, allowing the attacker to execute code on the victim's machine. This is typically performed through the use of JavaScript code contained in the file [15]. The PDF itself is not malicious in phishing attacks, but it attempts to persuade the user to click on a malicious link. Such efforts have just lately been uncovered and are, by definition, much more difficult to detect [7]. Every year, Adobe Reader, the most popular software for viewing PDF files, is updated to fix dozens of vulnerabilities in PDF documents [7].

JavaScript in PDF documents provides functionality and makes the document interactive. Forms are an example of an extra feature, which often comprise checkboxes, text inputs, action buttons, and other components that give the page with application-like behavior [26]. According to [26] since Acrobat Reader introduced JavaScript capabilities, even before the PDF format was included as an Open Standard, making of JavaScript in PDF files to perform malicious operations has been increasingly widespread.The hostile JavaScript code placed in the document is sometimes just a means for downloading and executing a specific instance of malware, or even a helpful code to exploit a specific weakness in the reader, [26].

Although JavaScript can be disabled in Adobe Reader since at least version 6, the risks of JavaScript code included in PDF format have historically been advised. Exemplary PDF files showing JavaScript flaws were supplied by David Kierznowski. And even without the use of JavaScript andwithout depending on

anything other than the PDF standards itself, Didier Stevens was able to execute malicious script from a PDF file read by Adobe Reader in 2010 [26].

There are numerous approaches for detecting malicious documents [24] and despite recent advances in machine learning for malware detection, antivirus companies continue to rely heavily on handwritten signatures to detect malicious PDFs [4]. This not only necessitates a large number of human resources, but it is also rarely effective in detecting unknown variants or zero-day attacks. Another option is to run the files for dynamic analysis in a managed virtualized environment. While this method generally increases the likelihood of identifying potential malware, it also takes a long time and requires access to a sandbox virtual environment. It also requires a human to define the detection rules based on file behavior [7]. Another popular alternative to signature analysis is dynamic document analysis, which involves observing the activity of the overall system or collection of applications while the document is being accessed [24].

Sophisticated document-based malware assaults that target user weaknesses in document-viewing client software or document structure are growing more widespread, and as the number of people who utilize PDF files increases, so does the need to restrict the repercussions of these malware-bearing documents. This study presents a framework for detecting and predicting malware-infected or malware-bearing PDF documents using Support Vector Machines (SVM). Using the Support Vector Classifier and preprocessing of PDF files, this study aims for 99 percent accuracy with a much lower False Positive Rate (FPR). Using this model, we present a mobile application that can classify any PDF file as benign or malicious based on its features.

2. Literature Review

PDF Documents harboring malicious payload have been the topic of much investigation throughout the years. Many ways for malware detection in PDF documents have been described in recent literature; past work on the detection of document malware shares many ideas with the methods for detecting malware in documents. One of the most prevalent is the use of machine learning approaches to enhance malware detection based on certain properties of the document itself. The advantage of using machine learning is that it is particularly effective against novel attacks or malware families that have never been seen before, as compared to traditional methods such as signature-based systems, which are mostly effective against known malware.

Early static methods relying on n-gram collection and analysis of document content by [10] and [23] were never tested against modern PDF malware. The scope of experimental evaluation in this work was relatively minimal. It included malicious PDF documents created by the attacker as well as a small number of samples (less than 300) from the out-of-date VXHeavens malware repository. They are

believed to be inadequate since they do not address several essential properties of the PDF format, such as encoding, compression, and encryption, and may be readily circumvented by contemporary PDF malware employing tactics similar to those employed against traditional signature-based antivirus systems.
[25] demonstrated that detecting malicious word files using static analysis, n-gram representation for document data, and dynamic analysis has great promise but is limited by the amount of malcode. Whilst technique and file type vary, the procedure's overall goals are quite similar to those of PDF malware detection.
Recent malware detection research has used dynamic analysis to collect characteristics, which implies that the features are retrieved while the binary is operating in a simulated environment. [17] used engineered reader software and dynamic analysis to collect structural characteristics from PDF texts for use in a machine-learning-based classifier, specifically Hellinger distance decision trees, because of their robustness in training data classing imbalances. Despite the fact that this is a high-level technique, they were unable to exhibit high detection rates since half of the malware test samples were classified as non-malware.
[27] conceived and implemented MDScan to overcome difficulties created by the static analysis technique. They integrated static and dynamic analysis for malicious PDF detection to improve detection. This feature, on the other hand, advances toward reliable detection of malicious PDF documents, although the emulation approach lengthens processing time.
The efficacy of detection while using machine learning approaches is based on numerous areas of the documents or tactics for collecting information that allow the algorithms to be much more effective. [12] suggested employing lexical analysis of JavaScript code as the input for the machine learning model in their technique (PJSCAN). For efficiency, the PJSCAN JavaScript extractor only looks for areas where the PDF standard specifies the presence of JavaScript. Moreover, if the JavaScript code has been designed to appear more like valid code, for example, lexical analysis of the code might not have been effective enough for malware identification if the malicious code is mixed with a large portion of genuine code, or if JavaScript code is placed in an arbitrary position accessible via the PDF JavaScript API and retrieved using an eval ()-like function call.
Another popular method for detecting malware in PDF documents is to use the document information as inputs for the feature extraction used during prediction. [24] developed their own extraction approach that depends heavily on the quality of a document's metadata in conjunction with the document's "structural properties." They collected 202 features that are closely connected to the document content, such as the number of pictures present, string size, and so on. Although using metadata is an innovative strategy, this metadata is frequently forged by the attacker. Therefore, it is critical to examine not just the conventional metadata in hazardous documents, but also the counterfeited metadata.

[13] were the first to apply logical structure to differentiate between malicious and benign PDF files. The PDF Standard defines logical structure as a high-level concept that organizes fundamental PDF building components into a functioning document. As a deep static technique, the [13] approach is less vulnerable to PDF obfuscation and physical structure falsification, which afflict other approaches such as the [12] approach. The results of their research reveal that harmful file features such as the inclusion of JavaScript and the minimum usage of harmless information may be properly deduced from their logical structure. Nonetheless, their technique's detection performance was seen to be unreliable in a real-world window experiment employing timestamped data. Its feature description created a blind area that attackers may exploit, as well as its large feature set resulted in a more memory-intensive learning classifier.

[19] provide an example of an implementation that takes this method to its logical end by using the PDF file structure as a detection vector. As a result, only attributes relating to how PDF was created, such as unreferenced objects, disguised streams, camouflaged filters, and so on, are considered. The basic idea behind this approach is that when the structural features of a PDF document have been altered, malware's existence in the document is quite likely. In this instance, the given system (O-Checker) does not use machine learning to assess whether or not the document is flawed, but instead relies on predefined criteria (which suggests malware).The disadvantage of this strategy is that it may fail to detect new PDF viruses and does not perform well on PDF from countries other than Japan, but it is a useful source of data for machine learning implementation.

[14] released Hidost, a machine learning-based malware detection system that is an expansion of [13], to solve the shortcomings of [13]. Hidost, unlike their prior work, is not just based on PDF malware detection but also works with a variety of file formats. This method achieves these goals by constructing a model of malicious and benign samples based on their logical structure and content. A practical experiment with periodic retraining over numerous months was evaluated on a real-world data set comprising 407,037 malware and 32,567 benign. Hidost is substantially less vulnerable to malware hidden in disguised areas of PDF files than its predecessor.

To improve static analysis of PDF documents, [26] extracted metadata features from documents such as the number of versions, size, edition time, author, and structural features such as the presence of possible anomalies in the inner structure of the file as well as features relating to the way the document and its components are organized. These extracted features are then tested on three different machine learning algorithms: Support Vector Machine (SVM), Random Forest (RF), and Neural Networks (in their case, Multi-layer Perceptron(MP)) to see which one performs the best. Multi-layer Perceptron and Random Forest performed the best, while Support Vector Machine performed the worst. Despite

their high-level approach, the proposed system may not be able to categorize PDF malware variations since the quantity of malicious and benign samples is limited.

Every day, more research on harmful PDF detection is conducted. [8] work is one of the most recent works in PDF malware detection, in which they propose a new method of detecting malicious actions in PDF by using Convolutional Neural Networks (CNN) built to take a byte series of non-executables as input and predict if the provided series contains harmful code or not.

Because it is difficult to detect malicious activities within papers, the threat of malicious documents is growing. As a result of the preceding instances of problems, the data for both the training and test data sets for this study were picked from public malware sources, meaning that this system would be taught using well-known malware samples and families. This situation involves focusing on the primary advantage of machine learning (namely, Support Vector Machine) applications, staying ahead of handwritten signature-based methods in the identification of emerging threats.

This study however, will focus on the issue of PDF malware evading detection. This study does not rely on metadata information such as authors, creation dates, and so on, which can be easily manipulated to prevent detection. However, in order to detect malicious samples, this research will primarily focus on structural aspects of PDF files.

3. Methodology

The PDF document type is widely used, and public data sets are freely availableThis study focuses on detecting fresh and genuine PDF malware by recognizing malicious PDF documents while utilizing machine learning approaches, namely Support Vector Machine.

In this study, common keyword strings in PDF documents are collected based on the number of times they appear in both malicious and benign PDF document samples to investigate how JavaScript code is exploited in reader programs. Important keyword strings are picked, such as /JS and /JavaScript, which indicate the presence of JavaScript, "/AA" and "/OpenAction", which signal an automated action to be executed instantly when the page or document is viewed, "/ObjStm", which counts the number of object streams, and so on.

Because most malware-infected PDF documents are known to contain JavaScript and a high likelihood of launching operations. To increase detection rate and reduce the chance of the detection system avoiding adversarial assaults, innocuous PDF files containing various JavaScript and JavaScript activities are gathered and retrieved in addition.

The keyword string features that were chosen are then utilized to train the Support Vector Machine algorithm, resulting in a model that can readily recognize and classify malicious PDF documents from benign ones.

3.1. **Framework.** The following described the framework.

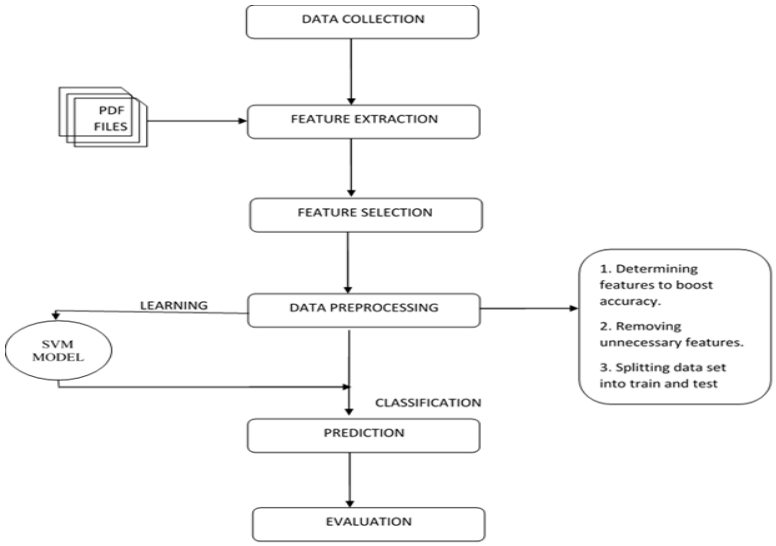


Figure 1: Proposed Framework

3.2. **Data Sources.** At this point, the study's goal is to collect a realistic data set that will allow the classifier to detect real-world malware PDF documents. For this investigation, PDF malware samples were gathered from a variety of sources and PDF document format versions. The malware samples obtained for this study include both old and new malware samples, with 70 percent of the malware samples being new.
In terms of the sources from which samples were gathered, this study chose them with the goal of representing the best feasible real-world scenario in mind. This study gathered samples from email accounts that typically receive spam or harmful attachments, as well as public malware repositories (VirusShare.com and Contagiodump). The Contagiodump repository contains curated malware and benign data sets targeted for signature research and testing, whereas VirusShare solely gives malware samples.

3.3. **Data Set.** This study specified two different classes or sets for sample kinds and classification:

3.3.1. *Benign.* These are samples with and without JavaScript retrieved from the trusted Contagiodump public document repository; the samples do not have any suspicious characteristics recognized by heuristics, such as obfuscation, suspicious API calls, and so on. The total number of samples is 10,000. This study used a set of antivirus vendors (through VirusTotal.com) to verify that the majority of the samples in this collection had not been recognized as malware by any vendor.

3.3.2. *Malicious.* This collection comprises a total of 10,000 malware samples gathered from the public sources Contagiodump and VirusShare. To guarantee that all obtained samples are malware (albeit some are not completely confirmed), they are checked on VirusTotal.com to ensure that the sample has been detected as malware by at least three anti-virus vendors.

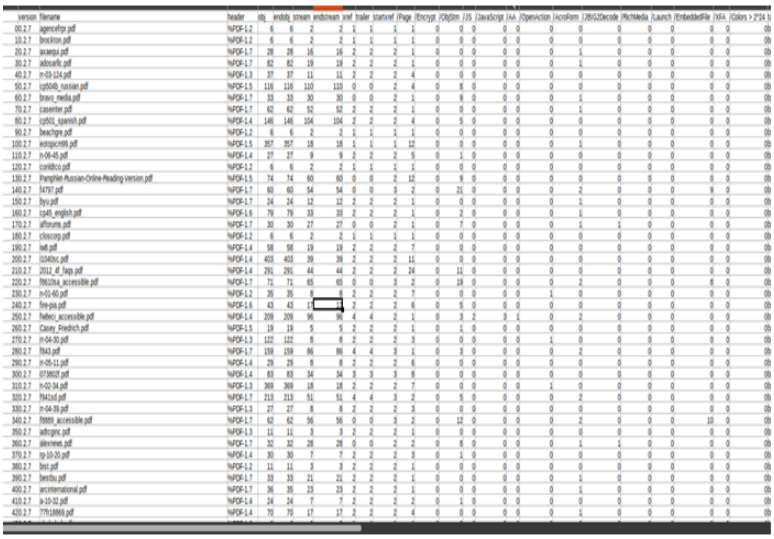


Figure 2: Benign PDF Files Features Dataset

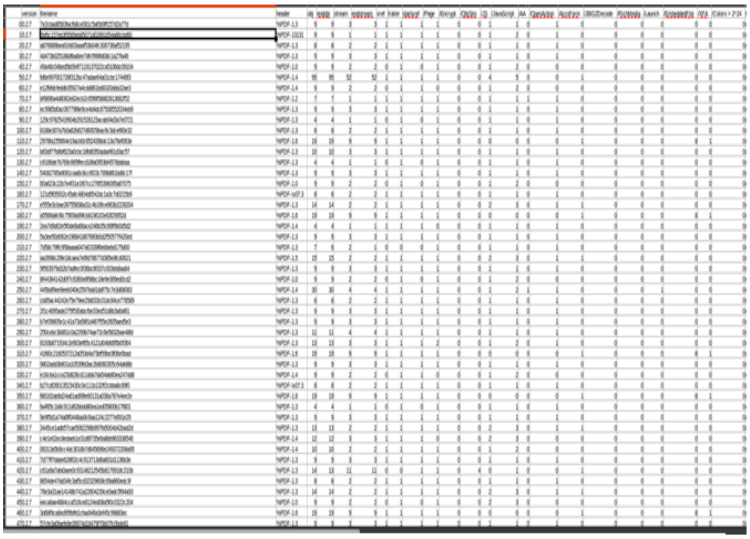


Figure 3: Malware PDF Files Features Dataset

3.4. **Feature Extraction.** There are various methods for extracting features from a PDF document. To achieve our goal, this study opted to capture structural features from PDF files using PDFiD. PDFid is a Python utility written by Didier Stevens to triage PDF documents; it can assist in distinguishing between PDF documents that may be malicious and those that are most likely not. Although PDFiD is not a parser, it searches for common keyword strings in PDF documents and counts the number of instances of the common keywords. Figure 3.4 demonstrates the use of PDFiD.

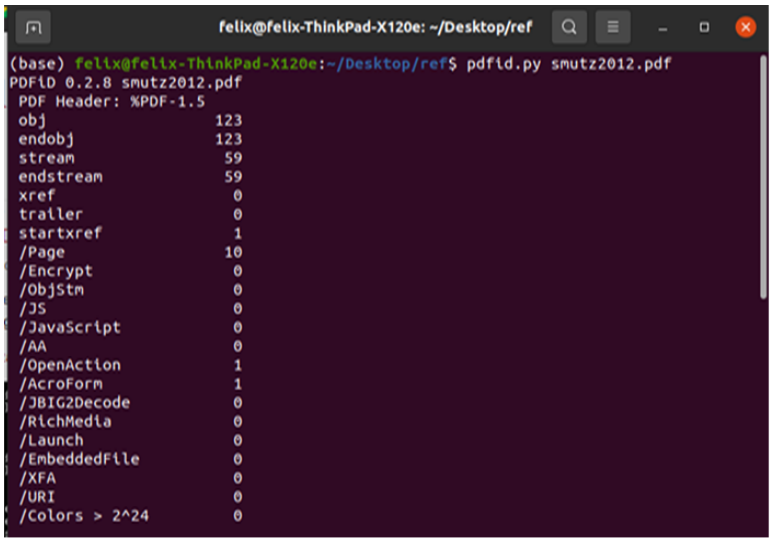


Figure 4: PDFiD Usage in Feature Extraction

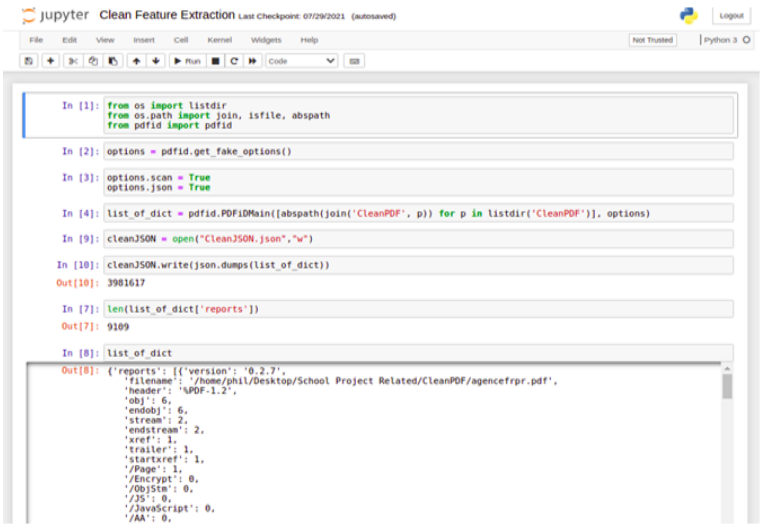


Figure 5: Extracting PDF Structural Features in Jupyter Notebook Environment

3.5. **Feature Selection.** The goal of this study is to deliver high classification quality and accuracy while maintaining a minimum False Positive Rate (FPR), a large number of retrieved features were chosen to be processed. This approach tries to be robust to variances in threats and vulnerabilities by concentrating on

patterns in the structural characteristics of PDF documents. As a consequence, the features chosen are intended to reduce dependency on specific strings from document metadata, such as author metadata, author field string length, and so on.

Similarly, keyword strings in structural features such as /Colors, filename, and so on are intentionally removed since they are seldom used; adding such features might result in better classification for known assaults but poor detection rates for fresh attack paths. To select the best features that will produce a good prediction, this study compares each feature with target variable to determine their level of importance. Figure 3.6 and 3.7 depicts some of the features compared to select important features where 0 represents benign and 1 represents malicious.

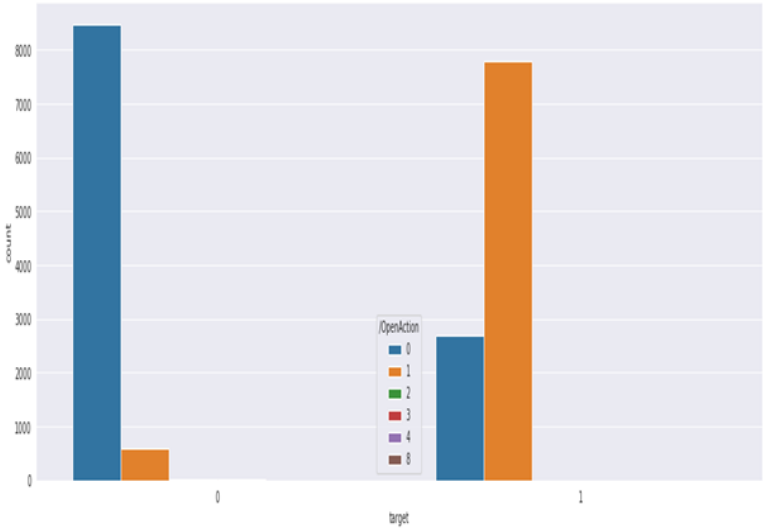


Figure 6: Relation Between JS and Target Variable

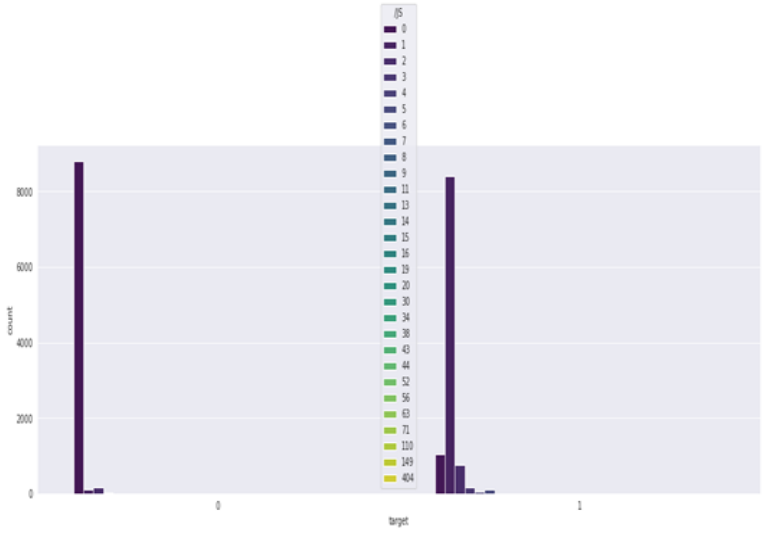


Figure 7: Relation Between OpenAction and Target Variable

3.6. **Classification Methodology.** The evaluated samples might be malicious or benign, this study utilized a supervised learning technique to address the problem as a binary classification problem. As a result, once the features were retrieved, selected, and pre-processed, this study separated the total number of recovered samples into two data sets: training and test.
After pre-processing the data set and removing null values, the total number of recovered samples is 19,601. The recovered samples were separated into training and test sets as follows:

(1) Training Set: consists of 70% of total recovered samples
(2) Test Set: consists of 30% of total recovered samples

Support Vector Machine (SVM) was chosen to optimize the sensitivity of the classifier in terms of accuracy. SVM works by accepting data points and generating a hyperplane known as the decision boundary. SVM was chosen for its high accuracy and lower computing complexity with a short execution time.

3.7. **Performance Metrics.** SVM's performance was evaluated using four distinct metrics.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$F1 - Score = \frac{Precision - Recall}{Precision + Recall} \tag{4}$$

Here, TP denotes True Positives, TN denotes True Negatives, FP denotes False Positives, and FN denotes False Negatives.

3.8. **Planning Stage.** Existing systems were investigated at this phase to gain understanding of the topic area. The similarities and variations between current systems were examined in order to identify the flaws in each. This phase also includes the configuration of the programming environment, the installation process, the selection of packages and libraries required for implementation, the learning of recommended programming techniques, and the communication of data between various libraries and languages.

3.8.1. *Development Tools.* This includes the programming languages used to create software that can identify malware in PDF documents. The computer system used during the implementation phase was a Lenovo ThinkPad X120e with 6GB RAM, 120GB hard drive, and 1.5GHz AMD processor. This research was carried out using the Linux operating system (Ubuntu 20.04 LTS).

3.8.2. *Programming Languages.* Python was used to carry out the machine learning experiment and deployment. Python is an open source programming language with large and extensive libraries that supports Object-Oriented, Function-Oriented, and Procedural programming paradigms. Python adheres to a core grammatical guideline that makes code more readable and application maintenance easier. This syntax allows you to communicate your thoughts without writing any more code. This project is built with Python and numerous of its libraries that are important to this issue.
JavaScript was used to create a client-side mobile application that uses the PDF malware classifier. JavaScript is a scripting language that enables complicated functionality to be implemented on web pages. Although JavaScript is well known for its web performance, it is now also utilized in creating cross-platform mobile applications with the help of a framework.

3.8.3. *Libraries and Frameworks used.* Pandas, Scikit-Learn, PDFiD, and Django are some of the most popular Python libraries used in this project. The React Native framework was used to create a client-side mobile PDF malware detecting application.
Pandas is a Python package that offers data structures that are rapid, adaptable, and expressive, making it easy and natural to deal with "relational" or "labeled"

data. Its objective is to serve as the basis for performing high-level actual data analysis in Python. Pandas Data Frames are a form of data structure that has been introduced to help with data manipulation. This tool facilitated viewing data, creating data, choosing or changing columns and indexes, rearranging data, and other tasks.
Sci-Kit Learn, also known as sklearn, is an open source Python library built on top of Numpy (another Python library that supports large mathematical functions to operate on arrays and matrices), SciPy (a Python library for scientific and technical computing), and matplotlib (a Python library for plotting data) (a data visualization library for Python). Sci-Kit Learn is a set of functions and classes that make Machine Learning and Deep Learning easier to use. Because the Support Vector Classifier is included in the Scki-Kit Learn package, Sci-Kit Learn was used for classification and data pre-processing in this study.

3.8.4. *PDFiD.* is a Python program for triaging PDF documents. It can help discern between PDF documents that may be harmful and those that aren't. Despite the fact that PDFiD is not a parser, it analyzes PDF documents for frequent keyword strings and counts how many times they appear.

3.8.5. *Django.* is an open-source Python-based web framework that adheres to the model–template–views design approach. In this study, the Django Framework is used to deploy the machine learning model on a web server, making it available from any device.

3.8.6. *React Native (or RN).* is a prominent JavaScript-based mobile app framework that allows you to develop native-looking iOS and Android apps. You may build applications for a range of platforms using the framework and the same codebase.

3.9. **Integrated Development Environment (IDE).** This research employed the Jupyter Notebook IDE and the Visual Studio Code text editor to achieve its goals and objectives. Jupyter Notebook is an open-source web application that allows you to interpret, edit, and exchange documents that contain scripts, graphics, and narrative prose. In this study, Jupyter Notebook was used for exploratory data analysis (EDA) and data visualization.
Visual Studio Code (also known as VSCode) is a text editor developed by Microsoft. Among the features are debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and an embedded Git for version control. This project's codebase was created in VSCode, a text editor.

4. Result

The Accuracy, Recall, Precision, and F1-Score metrics are used to evaluate the Support Vector Classifier's experimental results.

```
In [31]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

In [32]: svm = SVC()

In [33]: svm.fit(X_train, y_train)
Out[33]: SVC()

In [34]: predictions = svm.predict(X_test)

In [36]: print(accuracy_score(y_test, predictions))
         print('\n')
         print(confusion_matrix(y_test, predictions))
         print('\n')
         print(classification_report(y_test, predictions))

         0.8758714504335997


         [[2106  643]
          [  87 3845]]


                       precision    recall  f1-score   support

               benign       0.96      0.77      0.85      2749
            malicious       0.83      0.97      0.89      3132

             accuracy                           0.88      5881
            macro avg       0.89      0.87      0.87      5881
         weighted avg       0.89      0.88      0.87      5881
```

Figure 8: Result of SVM without hyper parameters

4.1. **Result of SVM without hyper parameters.** In the figure above, we can see that the model is failing to meet the study's objectives. Hyperparameters were searched with GridSearchCV to find the optimal parameters for the SVC in order to improve model accuracy.

```
In [2]: df_malware = pd.read_csv("NewMalwareCSV.csv", index_col=0)
        df_clean = pd.read_csv("CleanCSV.csv", index_col=0)

In [3]: df = pd.merge(df_malware, df_clean, how='outer')
        floated_header = pd.to_numeric(df['header'].str.replace('[^\d.]', ''), errors='coerce')
        df['header'] = floated_header

        <ipython-input-3-e7f52f54144f>:2: FutureWarning: The default value of regex will change from True to False in a fut
        ure version.
          floated_header = pd.to_numeric(df['header'].str.replace('[^\d.]', ''), errors='coerce')

In [4]: df.dropna(inplace=True)

In [5]: X = df.drop(['filename', '/Colors > 2^24', 'target', 'version'], axis=1)
        y = df['target']
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

In [7]: param_grid = {'C': [0.1,1,10,100, 1000], 'gamma': [1,0.1, 0.01, 0.001, 0.0001]}
        grid = GridSearchCV(SVC(), param_grid=param_grid, verbose=3)
        grid.fit(X_train, y_train)

        ...

In [8]: grid.best_params_
Out[8]: {'C': 1000, 'gamma': 0.001}
```

Figure 9: Using GridSearchCV to Search for Best Estimators

4.2. **Results of SVM with Hyper Parameters.** The SVM performs effectively using hyper parameters obtained with GridSearchCV, as can be shown in the above result.

```
In [10]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

In [11]: svm = SVC(C=1000, gamma=0.001)

In [12]: svm.fit(X_train, y_train)
Out[12]: SVC(C=1000, gamma=0.001)

In [13]: predictions = svm.predict(X_test)

In [14]: print(accuracy_score(y_test, predictions))
         print('\n')
         print(confusion_matrix(y_test, predictions))
         print('\n')
         print(classification_report(y_test, predictions))

         0.9982996089100493


         [[2745    4]
          [   6 3126]]


                       precision    recall  f1-score   support

               benign       1.00      1.00      1.00      2749
            malicious       1.00      1.00      1.00      3132

             accuracy                           1.00      5881
            macro avg       1.00      1.00      1.00      5881
         weighted avg       1.00      1.00      1.00      5881
```

Figure 10: Results of SVM with Hyper Parameters

```
# using SVM
svc = SVC(C=1000, gamma=0.001)
svc.fit(X_train, y_train)


# Storing model in pickle for faster use
pickle.dump(svc, open("svm_model.sav", "wb"))
pickle.dump(sc_x, open("standard_scaler.sav", "wb"))
```

Figure 11: Dumping Model into Memory for Faster Access on the Webserver

Model optimization and model deployment on a Django App were investigated in the figures above. The outcome of the machine learning process was dumped

into a pickle (memory) and also loaded into the memory, as can be seen. The model's capacity to predict was available via an Application Program Interface (API) so that it could be used on a variety of devices.

Output Design



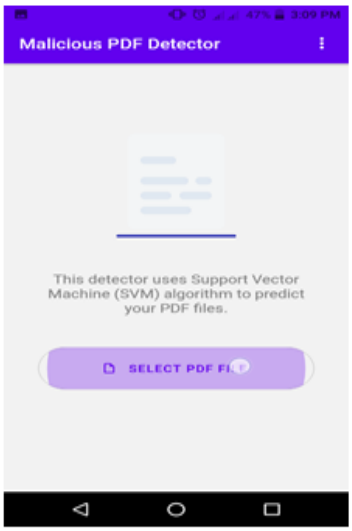


Figure 12: Welcome Screen
Figure 13: Main Screen

Conclusions: This study successfully shows that by extracting a broad feature set rather than depending on metadata information, a robust malware classifier can be built which produces high True Positives (TP) while keeping a low False Positive rate (FP). The results of 10,000 malicious and benign samples yielded a classification rate of over 99 percent, with low false positive rates of less than 0.2 percent. By applying the model to the features of the new variant data, it was also proved that the classifier is effective in identifying new versions of harmful PDF. The result demonstrates that classification is based on a vast number of characteristics, making escape much more difficult. The study also recognized that eliminating metadata features of PDF files, the files could be easily manipulated by attackers and reduced the likelihood of detection evasion and also make the model more resilient against mimic attacks. It is therefore recommended that the produced system should be used by individuals and organization to predict their PDF files before making use of the them.

Acknowledgement. The authors are grateful to University of Ilorin for the supports they received during the compilation of this work.

Competing interests: The manuscript was read and approved by all the authors. They therefore declare that there is no conflicts of interest.

Funding: The authors received no financial support for the research, authorship, and/or publication of this article.

References

[1] Acosta-Vargas P., Luján-Mora S. & Acosta T. (2017). Accessibility of Portable Document Format in Education Repositories. *Proceedings of the 2017 9th International Conference on Education Technology and Computers- ICETC.* **2017**, 239 pages.

[2] Alperovitch D. & Firm M. (2011). "Revealed: operation shady RAT," *An investigation of targeted intrusions into more than 70 global companies, governments, and non-profit organizations during the last five years.* pp. 1-2.

[3] Arjun Guha, ClaudiuSaftoiu, and ShriramKrishnamurthi (2010) "The Essence of JavaScript," *European conference on Object-oriented programming, 2010, pp. 126-150.*

[4] Beuhring, A., & Salous, K. (2014), "Beyond Blacklisting: Cyberdefense in the Era of Advanced Persistent Threats," *IEEE Security & Privacy, 12*(**??**)*, pp.90–93.*

[5] Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992), "A training algorithm for optimal margin classifiers," *Proceedings of the Fifth Annual Workshop on Computational Learning Theory – COLT 1992*, pp.144-152.

[6] Corona, I., Maiorca, D., Ariu, D., & Giacinto, G. (2014), "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 - AISec 2014*, pp.47-57.

[7] Fetteya, R., & Mansour R. (2020), "Detecting malicious PDF using CNN, " Conference Program Chairs - *ICLR 2020.*

[8] Jeong, Y.-S., Woo, J., & Kang, A. R. (2019), "Malware Detection on Byte Streams of PDF Files Using Convolutional Neural Networks," *Security and Communication Networks, 2019*, pp.1–9.

[9] Jordan, M. I., & Mitchell, T. M. (2015), "Machine learning: Trends, perspectives, and prospects. Science," *349(6245)*, pp.255–260.

[10] Li, W. J., Stolfo, S., Stavrou, A., Androulaki, E., & Keromytis, A. D. (2007), "A Study of Malcode-Bearing Documents," *Lecture Notes in Computer Science*, pp.231–250.

[11] Kramer, S., & Bradfield, J. C. (2009). "A general definition of malware". *Journal in Computer Virology, 6*(**??**), pp.105–114.

[12] Laskov, P., &Šrndić, N. (2011), "Static detection of malicious JavaScript-bearing PDF documents," *Proceedings of the 27th Annual Computer Security Applications Conference on - ACSAC 2011,* pp.373-382.

[13] Laskov, P. and Šrndić, N. (2013), "Detection of Malicious PDF Files Based on Hierarchical Document Structure," *Proceedings of the 20th Annual Network & Distributed System Security Symposium - NDSS 2013.*

[14] Laskov, P. and Srndic, N. (2016), "Hidost: a static machine-learning-based detector of malicious files," *EURASIP Journal on Information Security volume 2016.*

[15] Maiorca, D., Biggio, B., & Giacinto, G. (2019).“Towards adversarial malware detection: Lessons learned from PDF-based attacks,” *ACM Computing Surveys (CSUR),* pp. 1-36.

[16] Mell, P., Kent, K., & Nusbaum, J. (2005),“Guide to malware incident prevention and handling,” *Gaithersburg, Maryland: US Department of Commerce, Technology Administration, National Institute of Standards and Technology*, pp 800-83.

[17] Munson, M.A. and Cross, J.S. (2011), “Deep PDF parsing to extract features for detecting embedded malware,” *United States*.

[18] Orozco-Arias, S., Piña, J. S., Tabares-Soto, R., Castillo-Ossa, L. F., Guyot, R., & Isaza, G. (2020), “Measuring Performance Metrics of Machine Learning Algorithms for Detecting and Classifying Transposable Elements,” *Processes*.

[19] Otsubo Y, Mimura M, Tanaka H. (2016), “O-checker: Detection of Malicious Documents through Deviation from File Format Specifications,” *Black Hat USA 2016*.

[20] Santos, I., Devesa, J., Brezo, F., Nieves, J., & Bringas, P. G. (2013). OPEM: A Static-Dynamic Approach for Machine-Learning-Based Malware Detection,” *International Joint Conference CISIS'12-ICEUTE´12-SOCO 2012 Special Sessions*, pp.271–280.

[21] Selvaraj, K. and Gutierrez (2010), “The Rise of PDF Malware,” *Symantec Security Response 2010*

[22] Singh, P., Tapaswi, S., & Gupta, S. (2020). ”Malware Detection in PDF and Office Documents: A survey. Information Security Journal,” *A Global Perspective*, pp.1–20.

[23] Shafiq, M. Z., Khayam, S. A., & Farooq, M. (2008), “Embedded Malware Detection Using Markov n-Grams,” *Lecture Notes in Computer Science*, pp.88–107.

[24] Smutz, C., & Stavrou, A. (2012), “Malicious PDF detection using metadata and structural features,” *Proceedings of the 28th Annual Computer Security Applications Conference on - ACSAC 2012*.

[25] Tabish, S. M., Shafiq, M. Z., & Farooq, M. (2009) Malware detection using statistical analysis of byte-level file content. *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics - CSI-KDD 2009*, pp. 23-31.

[26] Torres J. and Santos S. D. L. (2018), ”Malicious PDF Documents Detection using Machine Learning Techniques,” *Proceedings of the 4th International Conference on InformationSystemsSecurity and Privacy - ICISSP 2018*, pp.337-344.

[27] Tzermias, Z., Sykiotakis, G., Polychronakis, M., & Markatos, E. P. (2011), “Combining static and dynamic analysis for the detection of malicious documents,” *Proceedings of the Fourth European Workshop on System Security - EUROSEC 2011.*

[28] Wagner, C., Wagener, G., State, R., & Engel, T. (2009), “Malware analysis with graph kernels and support vector machines,” *2009 4th International Conference on Malicious and Unwanted Software (MALWARE)*. pp. 63-68.